Efficient Architectures for Multidimensional Discrete Transforms in Image and Video Processing Applications

By

Saad Mohammed Saleh Al-Azawi

School of Electrical and Electronic Engineering

A thesis submitted for the degree of
Doctor of Philosophy

Faculty of Science, Agriculture and Engineering
Newcastle University, June 2013

Declaration

I declare that this thesis is my own work and it has not been previously submitted, either by me or by anyone else, for a degree or diploma at any educational institute, school or university. To the best of my knowledge, this thesis does not contain any previously published work, except where another person's work used has been cited and included in the list of references.

Saad Mohammed Saleh Al-Azawi

Supervisor’s Certificate

This is to certify that the entitled thesis **“Efficient Architectures for Multidimensional Discrete Transforms in Image and Video Processing Applications”** has been prepared under my supervision at the school of Electrical and Electronic Engineering /Newcastle University for the degree of PhD in Electronic Engineering.

Signature

Supervisor: Professor Said Boussakta

Date: June 2013

Signature

Student: Saad Mohammed Saleh Al-Azawi

Date: June 2013

To My Beloved Family

Acknowledgements

In the name of Allah, the Beneficent and the Merciful. Praise and Gratitude be to Allah for giving me strength and guidance, so that this thesis can be finished accordingly.

I would like to thank my supervisors: Professor Said Boussakta and Professor Alex Yakovlev. Please let me express my deep sense of gratitude and appreciation to both of you for the knowledge, guidance and unconditional support you have given me. I wish you all the best and further success and achievements in your life.

My deepest gratitude goes to my dearest parents, for their immense patience and unconditional support and encouragement throughout my life. My brothers, sisters and their daughters and sons: thank you very much for your prayers and encouragements. My friends and colleagues: thank you very much for what you have done for me. I thank you all for the companionship that has made this journey much easier. In fact, I do not need to list your names because I am sure that you know who you are.

I would like to express my sincere gratitude to Dr. Omar Nibouche for the very useful feedback on my research work. Ms. Sharon Pointer, thank you very much for your useful English language advice and comments. Also, I would like to thank all the Electrical and Electronic Engineering School and Newcastle University staff for their support during my study.

Finally, I also thank the Iraqi Ministry of Higher Education and Scientific Research, the Iraqi Cultural Attaché in London, Diyala University and the College of Engineering/Diyala University for supporting me during my study abroad.

Abstract

This thesis introduces new image compression algorithms, their related architectures and data transforms architectures. The proposed architectures consider the current hardware architectures concerns, such as power consumption, hardware usage, memory requirement, computation time and output accuracy. These concerns and problems are crucial in multidimensional image and video processing applications.

This research is divided into three image and video processing related topics: low complexity non-transform-based image compression algorithms and their architectures, architectures for multidimensional Discrete Cosine Transform (DCT); and architectures for multidimensional Discrete Wavelet Transform (DWT). The proposed architectures are parameterised in terms of wordlength, pipelining and input data size. Taking such parameterisation into account, efficient non-transform based and low complexity image compression algorithms for better rate distortion performance are proposed. The proposed algorithms are based on the Adaptive Quantisation Coding (AQC) algorithm, and they achieve a controllable output bit rate and accuracy by considering the intensity variation of each image block. Their high speed, low hardware usage and low power consumption architectures are also introduced and implemented on Xilinx devices.

Furthermore, efficient hardware architectures for multidimensional DCT based on the 1-D DCT Radix-2 and 3-D DCT Vector Radix (3-D DCT VR) fast algorithms have been proposed. These architectures attain fast and accurate 3-D DCT computation and provide high processing speed and power consumption reduction. In addition, this research also introduces two low hardware usage 3-D DCT VR architectures. Such architectures perform the computation of butterfly and post addition stages without using block memory for data transposition, which in turn reduces the hardware usage and improves the performance of the proposed architectures.

Moreover, parallel and multiplierless lifting-based architectures for the 1-D, 2-D and 3-D Cohen-Daubechies-Feauveau 9/7 (CDF 9/7) DWT computation are also introduced. The presented architectures represent an efficient multiplierless and low memory requirement CDF 9/7 DWT computation scheme using the separable approach.

Furthermore, the proposed architectures have been implemented and tested using Xilinx FPGA devices. The evaluation results have revealed that a speed of up to 315 MHz can be achieved in the proposed AQC-based architectures. Further, a speed of up to 330 MHz and low utilisation rate of 722 to 1235 can be achieved in the proposed 3-D DCT VR architectures. In addition, in the proposed 3-D DWT architecture, the computation time of 3-D DWT for data size of 144×176×8-pixel is less than 0.33 ms. Also, a power consumption of 102 mW at 50 MHz clock frequency using 256×256-pixel frame size is achieved. The accuracy tests for all architectures have revealed that a PSNR of infinite can be attained.

Contents

List of Figures

List of Tables

Abbreviations

Accum	Accumulators
Adds_Generator	Address Generator
AMBTC	Absolute Moment Block Truncation Coding
AQC	Adaptive Quantisation Coding
ARM	Advanced RISC Machine
ASIC	Application Specific Integrated Circuits
bior	biorthogonal wavelet
bpp	bit per pixel
BRO	Bit Reverse Order operation
BTC	Block Truncation Coding
BTFL	Butterfly
CDF 9/7	Cohen-Daubechies-Feauveau (9/7)
CLB	Configurable Logic Block
CMOS	Complementary Metal Oxide Semiconductors
CMux	Column Multiplexer
Comp	Comparator
C_t	Computation Time
CT	Cosine transform
Ctrl	Controller signal
CU	Column unit
Daub-4	Daubechies 4-tap filter or CDF 5/3 or LeGall5/3
DCT	Discrete Cosine Transform
D-FF	D-Flip Flop
DFT	Discrete Fourier Transform
DSP	Digital Signal Processing

DWT	Discrete Wavelet Transform
EBCOT	Embedded Block Coding with Optimized Truncation
EDF	Error Diffusion
EDK	Embedded Development Kit
EZW	Embedded Zerotree Wavelet
FF	Flip Flop
FIFO	First In First Out
FIR	Finite Impulse Response
FMux	Frame Multiplexer
FPGA	Field programmable Gate Arrays
FT	Fourier Transform
FU	Frame Unit
FWL	Floating Wordlength (mantissa part)
GOF	Group Of Frames
HLS	High Level Synthesis
HPF	High Pass Filter
HVS	Human Visual System
I/O	Input/Output
IBAQC	Intensity Based Adaptive Quantisation Coding
IDCT	Inverse Discrete Cosine Transform
IHPF	Inverse High Pass Filter
ILPF	Inverse Low Pass Filter
IP	Intellectual Property
ISE	Integrated Software Environment
IWL	Integer Wordlength
JPEG	Joint Photographic Expert Group
Kb	Kilo bits

LCD	Liquid Crystal Display
LE	Logic Element
LPF	Low Pass Filter
LSB	Least Significant Bit
LUT	Look Up Table
MA	Multiplier-Adder
Mem_Enb	Memory Enabling
MF	Maximum operating Frequency
MHz	Mega Hertz
ms	millie second
mW	millie Watt
MPEG	Moving Picture Expert Group
MRC	Memory Reading Controller
MRE	Memory Reading Enable
MRI	Magnetic Resonance Imaging
MSB	Most Significant Bit
Mult	Multiplier
Mux	Multiplexer
MWC	Memory Writing Controller
NormF	Normalisation Factor
PSNR	Peak Signal to Noise Ratio
R_adds	Reading address
R_enb	Reading enable
RAM	Random Access Memory
RC	Row Column
RCF	Row Column Frame
RCU	Row Column Unit

RISC	Reduced Instruction Set Computing
RMSE	Root Mean Square Error
ROM	Read Only Memory
RTL	Register Transfer Level
RU	Row Unit
SDP	Slice Delay product
SelC	Column Selector
SelF	Frame Selector
SelR	Row Selector
SPECK	Set Partitioned Embedded BloCk
SPIHT	Set Partitioning In Hierarchical Trees
SR	Shift Register
SRL	Shift Register LUT
SU	Single Unit
SubStg	Sub Stage
TF	Twiddle Factor
TMem	Transpose memory
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
VLSI	Very Large Scale Integration
VR	Vector Radix
W_adds	Writing address
W_enb	Writing enable
WL	WordLength
YC_bC_r	Luminance-Chrominance colour space

Chapter 1: Introduction

1.1 Introduction

Transforms, such as the Fast Fourier Transform (FFT), Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT), represent the main part of many audio, image and video processing systems. Although these transforms improve the performance of the entire system, they represent a bottleneck in terms of their computation complexity, storage requirements and subsequent processing speed and power consumption. These issues are crucial in multidimensional applications which require a high computation load and large amount of high speed storage media. As a result of these concerns, the idea and the subject of this thesis arose. This thesis presents new and efficient hardware architectures for low complexity non-transform based image compression algorithms. Furthermore, architectures for the transforms used in transform-based image and video processing systems, such as the DCT and the DWT, are also proposed. In addition, comprehensive analyses are introduced in order to evaluate the performance of the proposed hardware architectures. The performance analysis includes the hardware usage, power consumption, processing speed and output accuracy.

In summary, this chapter introduces: in section 1.2, a general background about digital images and video signals is provided. In section 1.3, an overview of image and video compression algorithms, and their related transforms and architectures are also briefly introduced. The motivation, objectives and contributions of this research are introduced in sections 1.4, 1.5 and 1.6, respectively. The thesis outline is presented in section 1.7 and the outcome publications of this research are outlined in section 1.8.

1.2 Digital Image and Video Signals

The data size of any image is related to its bit rate, number of colour bands, and image size, and on the number of images or frames in the case of medical or video streams. Thus, high volume storage devices are required to handle such a large amount of data.

The data storage requirements and bit rate are important factors that affect the complexity of any image and video processing system, particularly in mobile devices. In such devices, the main challenges are power consumption, hardware usage, storage resources and processing speed. In many image and video applications, the signal bit rate can be adjusted using lossy or lossless compression algorithms. The lossy compression algorithms sacrifice a specific amount of data accuracy to obtain high compression ratios and low bit rates, while in lossless compression algorithms; the decompressed data is exactly the same as the original, though with low compression ratios and high bit rates. For both compression algorithm classes, the computation complexity and storage of intermediate data represent an important factor that contributes to the complexity of the entire system. Thus, work has been conducted to achieve compression algorithms that produce good image quality, a low bit rate and low complexity in terms of storage requirements, high speed, low power consumption and low hardware usage.

1.3 An Overview of Image and Video Compression Algorithms

Numerous image and video compression algorithms have been published in the literature. This includes a class of non-transform based low complex algorithms such as Block Truncation Coding (BTC) and Adaptive Quantisation Coding (AQC) [1-5]. Further algorithms are classified as transform-based algorithms, as in the Joint Photographic Experts Group (JPEG) and JPEG2000 image compression standards [6, 7], and MPEGx and H.26x video compression standards [8]. These algorithms offer a good compromise between image quality and the output bit rate, where the bit rate and the output image quality can be adjusted by proper selection of the quality factor. Although the first class of image compression algorithms represents a low complexity and high speed compression systems, the low compression ratio represents its main drawback compared with the second class of algorithms. However, it represents a suitable choice for some applications that require low complexity and low hardware usage [9].

The second class of compression systems encompasses algorithms based on transforms such as the DCT and DWT. This increases the computation complexity, power consumption and hardware usage of such systems. To tackle these issues, many hardware architectures have been published in the literature for 1-D and 2-D DCT, and DWT [10-23], while a limited number of architectures have been suggested for 3-D DCT and DWT applications [22, 24-30]. Nevertheless, the suggested 3-D DCT and 3-D DWT in the literature have only partially met the requirements of hardware architectures such as low power consumption, high processing speed, low hardware usage and low memory requirements. As such, it is important to produce efficient architectures for the DCT and the DWT which consider such issues for multidimensional applications.

1.4 Research Motivation

The data storage, processing and transmission represent important parts of the major concerns and interest of the modern life. As an example, medical images and the related archiving systems require a high data storage capacity to handle the information of every patient. Additionally, the collected information is usually processed in different ways to prepare it for storage and transmission media to overcome their restrictions and challenge. Considering these points and other concerns, the data compression and the tools required in this field represent the backbone of these applications. However, the data processing is not a straightforward task as it has its own restrictions and challenges. Among these challenges is the computation complexity, processing time, power consumption and storage area. This thesis acts on the previous issues to provide efficient hardware architectures for the data transforms used in image and video compression systems, especially when these issues become crucial in the case of multidimensional signals, such as medical images and video streams.

1.5 Research Objectives

The research aims are to introduce new low complexity image compression algorithms, their corresponding architectures and efficient architectures for multidimensional data transforms used in image and video processing systems. The proposed architectures focus on achieving high computation speed, low power consumption and parameterisation for different wordlengths. The objectives of this study can be outlined as follows:

- To present parameterised architectures for low complexity image compression algorithms and multidimensional data transforms.

- To introduce new and efficient 2-D non-transform-based and low complexity image compression algorithms and their hardware architectures for low bit rate applications. The proposed algorithms should consider the intensity variation of image blocks for further bit rate reduction while preserving the output image quality within an acceptable range.

- To design and implement new high speed multidimensional DCT architectures for image and video processing applications. Such architectures can be based on the available low computational complexity fast DCT algorithms.

- To design and implement new low hardware usage architectures for 3-D DCT to achieve a compromise between the hardware usage, processing speed and power consumption for variant wordlengths. The proposed architectures should consider hardware and memory usage reduction by reducing the data transposition throughout the computation process.

- To design and implement new hardware architectures for a lifting-based 1-D, 2-D and 3-D DWT to attain high throughput and low memory requirements for different wordlengths. In such architectures, the main goals are: multiplierless, high throughput, low memory requirements and low power consumption. In such architectures, the power consumption reduction may be achieved by reduce or avoid using the high power consumption elements, such as multipliers.

1.6 Contributions

The contributions of this thesis can be summarised as follows:

1. It introduces low complexity non-transform-based image compression algorithms for low bit rate applications. Their new high speed and low hardware usage architectures are also introduced.
2. It presents new high speed multidimensional DCT architectures, based on fast DCT algorithms for image and video processing applications.
3. It introduces new area efficient architectures for 3-D DCT computation using 3-D DCT VR algorithm.
4. It introduces parallel and multiplierless multidimensional DWT architectures to attain a good compromise between memory requirements, processing speed and power consumption.

1.7 Thesis Outline

The remainder of this thesis is organised as follows: Chapter 2 is an overview of image and video compression algorithms and their related data transforms, such as the DCT and DWT, which are used in 2-D image and 3-D video and, hyperspectral and medical image processing systems. It also introduces Field Programmable Gate Arrays (FPGA) devices as a medium for hardware implementation. An introduction to hardware design tools and Xilinx high level synthesis design tools is also provided.

In Chapter 3, new non-transform based and low complexity image compression algorithms for low bit rate applications are introduced. These algorithms are based on the AQC algorithm and they consider the intensity variation of each individual image block to introduce low-bit rate and high-performance compression systems. Moreover, new high speed architectures to implement these algorithms are also proposed. Furthermore, a comprehensive analysis of power consumption, speed and hardware resources is also provided.

Two high speed architectures for multidimensional DCT computation are introduced in chapter 4. These architectures are based on 1-D DCT Radix-2 and 3-D DCT VR fast algorithms. The proposed architectures are parameterised in terms of wordlength, which provides different levels of output accuracy, hardware usage and power consumption.

In chapter 5, two new low hardware usage architectures for the 3-D DCT VR algorithm are presented. The main feature of the proposed architectures are; the in place computation of the butterfly and post addition stages without using block memory for data transposition, which in turn produces efficient architectures in terms of hardware usage, processing speed and power consumption.

Multiplierless and parallel lifting-based architectures for the 1-D, 2-D and 3-D Cohen-Daubechies-Feauveau 9/7 (CDF 9/7) DWT are introduced in chapter 6. The proposed architectures represent an efficient low memory requirement, multiplierless and high precision CDF 9/7 DWT systems. In such architectures, the temporal memory for 3-D DWT has been reduced to a block memory size of four frames only. Such block memory is used as a temporary frame buffer during the computation process of the 3-D DWT. Conclusions and the outline for potential future work directions are given in chapter 7.

1.8 List of Publications

During the period of PhD research the following publications have been introduced:

- S. Al-Azawi, S. Boussakta, and A. Yakovlev, "Image Compression Algorithms Using Intensity Based Adaptive Quantisation Coding," American Journal of Engineering and Applied Sciences, vol. 4, pp. 504-512, 2011.
- S. Al-Azawi, S. Boussakta, and A. Yakovlev, "High precision and low power DCT architectures for image compression applications," in the IET Conference on Image Processing (IPR 2012), 2012, pp. 1-6.
- S. Al-Azawi, S. Boussakta, and A. Yakovlev, "Performance improvement algorithms for colour image compression using DWT and multilevel block truncation coding," in the 7th International Symposium on Communication Systems Networks and Digital Signal Processing (CSNDSP), 2010, pp. 811-815.
- S. Al-Azawi, S. Boussakta, and A. Yakovlev, "Low complexity image compression algorithm using AMBTC and bit plane squeezing," in 7th International Workshop on Systems, Signal Processing and their Applications (WOSSPA), 2011, pp. 131-134.

Chapter 2: Preliminary Concepts and Related Work

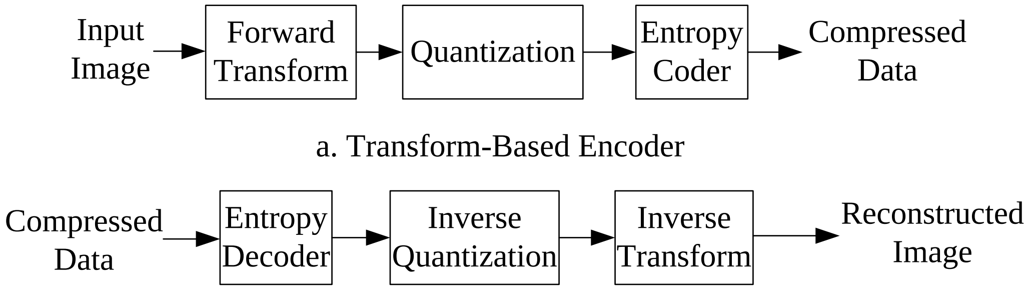
2.1 Introduction

This chapter presents explanatory materials that support the architectures presented in the remainder of the thesis. It is divided into six sections: section 2.2 is an overview of image/video compression algorithms and the associated transforms, including the DCT and DWT. The definitions of rate distortion measurements image fidelity criteria are given in section 2.3. Section 2.4 presents and overview about the related work. Section 2.5 introduces Xilinx FPGA hardware devices which are used as implementation media in the remainder of the thesis, with a particular emphasis on Virtex 5 and 6 Xilinx families given in section 2.6. Hardware design tools are presented in section 2.7. The architectures design procedure is described in 2.8 and the summary is given in section 2.9.

2.2 Image and Video Compression Algorithms

The revolution in data transmission and communication systems has made it of paramount importance to compress data (such as images and video) before storing or transmitting it; this is largely due to a limitation in bandwidth, bit rate and storage device size. As an example, a typical 512×512-pixel colour image requires a storage space of 786.432 Kb, if it has a typical bit rate of 8 bit per pixel per colour band (bpp). This number will increase in the case of medical or hyper-spectral images where the bit rate can be 16 or 24 bpp and, thus, in video streams the matter becomes more crucial due to the large number of images/frames and the size of each frame.

The idea of compression comes from reducing the redundancy of data which occurs from different types of correlation, such as correlation between neighbouring pixels in an image or frame, correlation between colour planes in colour images and correlation between pixels in the neighbouring frames. These three types of correlation are known as spatial redundancy, spectral redundancy and temporal redundancy, respectively [31]. These types of redundancies are classified as a statistical redundancy and psycho-visual redundancy. The later refers to the Human Visual System's (HVS) limitations [32]. Some algorithms exploit the spatial redundancy in two dimensional (2-D) image compression systems, while others algorithms exploits both spatial and temporal redundancy, which may be classified as a three dimensional (3-D) compression system. In another aspect, image compression can be classified as a lossless or lossy. The class of lossless compression algorithms ensures that the decompressed image is an exact version of the original image, this result in a low compression ratios or high bit rate. On the other hand, the lossy class of compression algorithms attains high compression ratios, which is nevertheless accompanied by a degradation of the decompressed image quality [33, 34]. JPEG and JPEG2000 image coding standards, as examples of transforms-based compression algorithms. The JPEG image compression standard is based on the DCT [6], and the JPEG2000 is a DWT-based compression system [7]. A typical transform-based compression system is shown in Figure 2.1 [7]. Even though the output performance of the transform-based compression algorithm is high, the computation complexity is their main drawback. The other class of compression algorithms is the low complex and good performance non-transform-based image compression, for which the BTC, multilevel BTC and AQC are well-known algorithms [1-3, 35]. Such algorithms are designed specifically to avoid or reduce computational complexity, memory requirement and processing time arising from applying the DCT or DWT.



b. Transform-Based Decoder

Figure 2.1: Basic block diagram of typical transform-based compression / decompression system [36].

2.2.1 *Adaptive Quantisation Coding (AQC)*

The AQC is a low complex image compression algorithm which represents an improved version of BTC algorithm [2, 35]. The BTC algorithm is used for grey-scale image compression for an output bit rate of two bit per pixel (bpp) and good image quality [1, 3]. The BTC partitions the input image into 4×4-pixel blocks and it computes a bit plane using a bi-level quantiser and two eight bit moments which represents the high and low mean of the image block. The low compression ratio represents the main drawback of the BTC when compared with other transformed–based image compression techniques. However, it is an attractive option for mobile and low complex applications due to its simplicity and good image quality [37]. Many researchers have tried to enhance the performance and to reduce the bit rate of BTC algorithm as stated in [4, 31, 37-42]. In [37] a three level quantiser was used to classify the pixels into low, intermediate and high intensity, according to a predefined threshold. Accordingly, the AQC algorithm partitions the input image into a fixed size blocks and it encodes the input image using a multilevel quantised bit plane, the minimum intensity value of the block and the quantiser step [4, 5]. Thus, the AQC can be considered as a multi-level BTC algorithm and the coding performance of the AQC algorithm is much better than the BTC but it comes in the expenses of higher bit rate. However, the bit rate remains the same for both algorithms for low intensity variation images. Thus, the AQC algorithm produces good compromises between coding performance and computation complexity. The efficient coding performance and low implementation cost of multilevel BTC and AQC algorithms have encouraged some researchers to use them as a suitable approach for memory usage reduction in Liquid Crystal Display (LCD) overdrive [5, 43-46] and in data hiding applications [3, 47-49].

2.2.2 Discrete Cosine Transform (DCT)

The DCT has gained a wide acceptance within the signal processing community since it was suggested in 1974 [50]. This is due to its energy compaction, orthogonal and performance level being close to that of Karhunen-Loeve transform with respect to the rate distortion criterion [50]. It has become an essential part of many image and video applications, including the JPEG, MPEGx and H.26x compression standards [6, 8, 51-53]. With DCT-based audio, image and video compression systems, the input signal is partitioned into N, $N \times N$-point data blocks or $N \times N \times N$-point data cubes, and then transformed into DCT domain using 1-D, 2-D and 3-D DCT transforms, respectively.

The 1-D DCT for N-point data can be computed as follows [16]:

$$X(k) = \frac{2\varepsilon}{N} \sum_{n=0}^{N-1} x(n) \times cos\left(\frac{\pi}{2N}(2n+1)k\right) \qquad (2.1)$$

where $k = 0, 1, 2 \ldots., N-1,\ \ n = 0,1,2\ \ldots\ldots.., N-1$ and $\varepsilon = \begin{cases} \frac{1}{\sqrt{2}}, & for\ k = 0 \\ 1, & otherwise \end{cases}$

The inverse 1-D DCT (1-D IDCT) is defined as:

$$x(n) = \sum_{k=0}^{N-1} \varepsilon \times X(k) \times cos\left(\frac{\pi}{2N}(2n+1)k\right) \tag{2.2}$$

The 2-D DCT of a $N \times N$-point data block can be computed by computing the 1-D DCT on the rows and columns successively. Alternatively, the 2-D DCT can be computed directly as [12]:

$$X(k_1, k_2) = \frac{\varepsilon_{k_1}\varepsilon_{k_2}}{N} \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1, n_2) \times cos\left(\frac{\pi}{2N}(2n_1+1)k_1\right) \times cos\left(\frac{\pi}{2N}(2n_2+1)k_2\right) \tag{2.3}$$

and the 2-D IDCT is computed as:

$$x(n_1, n_2) = \frac{2}{N} \sum_{k_1=0}^{N-1} \sum_{k_2=0}^{N-1} \varepsilon_{k_1}\varepsilon_{k_2} X(k_1, k_2) \times cos\left(\frac{\pi}{2N}(2n_1+1)k_1\right) \times cos\left(\frac{\pi}{2N}(2n_2+1)k_2\right) \tag{2.4}$$

where $k_1, k_2, n_1\ and\ n_2 = 0, 1, 2 \dots, N-1$ and $\varepsilon_{k_1} and\ \varepsilon_{k_2} = \begin{cases} \frac{1}{\sqrt{2}}, & for\ k_i = 0 \\ 1, & otherwise \end{cases}$

The 3-D DCT for an $N \times N \times N$-point data cube can be computed as follows [54]:

$$X(k_1, k_2, k_3) = \frac{8\varepsilon_{k_1}\varepsilon_{k_2}\varepsilon_{k_3}}{N^3} \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} \sum_{n_3=0}^{N-1} x(n_1, n_2, n_3) \times cos\left(\frac{\pi}{2N}(2n_1+1)k_1\right) \times cos\left(\frac{\pi}{2N}(2n_2+1)k_2\right) \times cos\left(\frac{\pi}{2N}(2n_3+1)k_3\right) \tag{2.5}$$

and the 3-D IDCT is given by [54]:

$$x(n_1, n_2, n_3) = \sum_{k_1=0}^{N-1} \sum_{k_2=0}^{N-1} \sum_{k_3=0}^{N-1} \varepsilon_{k_1}\varepsilon_{k_2}\varepsilon_{k_3} X(k_1, k_2, k_3) \times cos\left(\frac{\pi}{2N}(2n_1+1)k_1\right) \times cos\left(\frac{\pi}{2N}(2n_2+1)k_2\right) \times cos\left(\frac{\pi}{2N}(2n_3+1)k_3\right) \tag{2.6}$$

where $k_i\ and\ n_i = 0, 1, 2 \dots, N-1, i = 1,2,3$ and $\varepsilon_{k_i} = \begin{cases} \frac{1}{\sqrt{2}}, & for\ k_i = 0 \\ 1, & otherwise \end{cases}$

Further, the 3-D DCT can also be computed by applying N-point 1-D DCT using (2.1) on the rows then columns and the temporal or frames direction successively. This is termed the Row-Column-Frame (RCF).

It is obvious from (2.1) to (2.6) that the computation complexity of the DCT increases rapidly with the data dimensionality. As an example, the computation complexity of $N \times N \times N$-point data cube using an RCF approach requires $3N^2$ 1-D DCT, where each 1-D DCT requires N^2 multiplication and $N(N-1)$ addition operations. Nevertheless, the computation complexity of the 3-D DCT can be reduced using fast DCT algorithms. A number of fast DCT algorithms have been published in the literature, among them the 1-D Radix-2 algorithm [55] and 3-D DCT Vector Radix algorithm (3-D DCT VR) [54]. These fast algorithms reduce the mathematical operations involved in the computation of the DCT, thus, achieving shorter processing times. In addition they also reduce the memory space required for data transpose for both 2-D and 3-D DCT.

The arithmetic operations for the original RCF 3-D DCT, RCF 3-D DCT Radix-2 and 3-D DCT VR algorithms are shown in Table 2.1 [54, 56, 57]. It is obvious from Table 2.1 that the number of multiplications in a 3-D DCT VR algorithm is less than that required by a 3-D DCT Radix-2 algorithm by $\frac{5}{8}N^3 log_2N$.

Table 2.1: The arithmetic operations of the original RCF 3-D DCT, RCF Radix-2 and VR algorithms [54, 56, 57].

Operations	Original RCF 3-D DCT	RCF using 1-D DCT Radix-2	3-D DCT VR
Multiplication	$3N^4$	$\frac{3}{2}N^3 log_2N$	$\frac{7}{8}N^3 log_2N$
Addition	$3N^3(N-1)$	$\frac{9}{2}N^3 log_2N - 3N^3 + 3N^2$	$\frac{9}{2}N^3 log_2N - 3N^3 + 3N^2$

2.2.2.1 DCT-Based Compression Systems

A conventional DCT-based image compression system encompasses of: DCT, quantisation and encoding, as shown in Figure 2.2 [6].

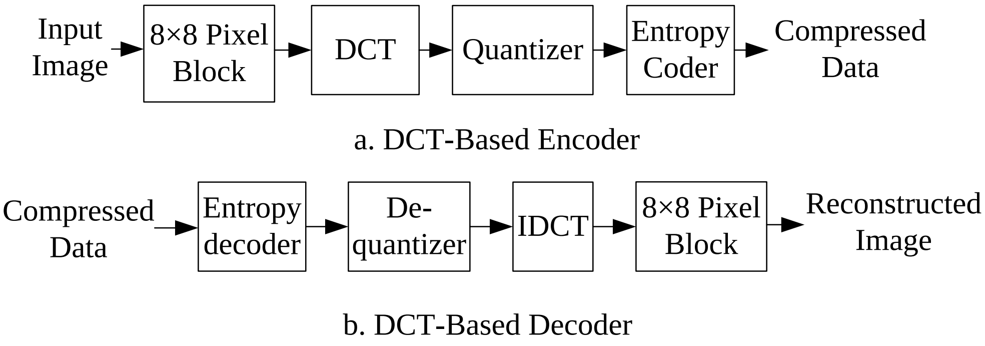


Figure 2.2: DCT based image compression/decompression system [36].

As shown in Figure 2.2, each 2-D input image is partitioned into fixed block sizes (typically 8×8-pixel), and then the DCT is applied to tackle the spatial correlation in the block. Furthermore, the quantisation and entropy coding are used to reduce the bit rate further.

In 3-D signals, such as videos and 3-D medical images, the spatial correlation between pixels can be reduced using the 2-D DCT, while the motion compensation process used to tackle the correlation between frames (temporal correlation). This makes the 3-D video compression systems, such as in MPEGx and H.26x standards, consist of 2-D compression system with a motion compensation unit. However, the high computation complexity of motion compensation makes it is difficult to get a real time 3-D compression system without using efficient hardware accelerators. Further, the motion estimation is computed with respect to a reference frame within a group of frames; this dependency can cause accumulated and propagated error throughout each group of frames [58, 59]. The computation complexity of the conventional 3-D images and videos has increased the need for new low complex compression system, particularly in mobile and wireless communication systems [58]. New approaches have been proposed in the literature that considers video data as a 3-D data volume. The correlation at the temporal direction is tackled by applying 1-D DCT along the third dimension while spatial data redundancy is considered using the 2-D DCT; this has led to new video and 3-D image compression systems using the 3-D DCT [24, 54, 59-63]. Such a transform is particularly suited for embedded systems requiring low complexity implementation of both video encoder and decoder, such as the case of mobile terminals with video-communication capabilities [24].

2.2.2.2 3-D DCT Applications

The 3-D DCT is used for different applications, such as 3-D video compression, hyper spectral image compression and watermarking [64-68]. In [64], an approach based on a JPEG lossy image compression system and a 3-D DCT for still image compression was introduced. The proposed system constructs an 8×8×8-pixel 3-D data cubes from the original 2-D image data using a spiral scanning. Such technique utilises a new quantisation table to improve the coding performance. It has been found that the performance of the proposed technique outperformed the conventional JPEG compression, especially at low and high bit rates in particular [64]. Furthermore, the 3-D DCT has been used to exploit the spatio-temporal redundancy in video watermarking [65-67] and hyper spectral image compression [68]. Other applications of 3-D DCT algorithms include object tracking [69], stereoscopic video quality assessment [70], 3-D video de-noising [71] and lip reading [72].

2.2.3 Discrete Wavelet Transform (DWT)

The DWT [73] is used in different image and video applications, such as compression, watermarking and denoising. Similar to the DCT, the DWT exhibits a high rate of energy compaction, thus making it a good choice in compression standards. Moreover, the DWT in image compression algorithms has the property to overcome the blocking artifact that characterise the DCT-based or block-based image compression techniques. Nevertheless, the DWT affected by its high computation complexity and memory requirements compared to the DCT or other block-based algorithms. Such requirements are dependent on the wavelet filter length and the size of input data.

Various low and high pass wavelet filters are used in DWT applications. The simplest wavelet filter is the Haar wavelet filter, which has the following coefficients for decomposition and reconstruction operations, respectively.

$$\text{LPF}=\left[\ 0.707 \quad 0.707\right] \text{ and } \text{HPF}=\left[-0.707 \quad 0.707\ \right] \qquad (2.7)$$

where LPF and HPF are the Low and High Pass Filter, respectively. The coefficients of Haar reconstruction filter are shown below:

$$\text{ILPF}=\left[\ 0.707 \quad 0.707\right] \text{ and } \text{IHPF}=\left[\ 0.707 \quad -0.707\right] \qquad (2.8)$$

where ILPF and IHPF are the Inverse Low and High Pass Filter, respectively.

Further, more sophisticated wavelets filters are available, which they are different in terms of output performance, filter order and computation complexity. As an example of these filters are Cohen-Daubechies-Feauveau 9/7 (CDF 9/7) and Le Gall 5/3 which are used in JPEG2000 lossy and lossless compression standards, respectively. The coefficients of both filters are shown in Table 2.2 [32, 74]. Moreover, there are infinite number of possible wavelet filters, such as Daubechies and Coiflets families, and the selection of the best wavelet filter depends on the required application [75, 76].

Table 2.2: CDF 9/7 and Le Gall 5/3 analysis filters coefficients [32, 74].

	CDF 9/7		Le Gall 5/3	
	LPF	HPF	LPF	HPF
0	0.852699	0.788486	1.060660	0.707107
±1	0.377402	0.418092	0.353553	0.353553
±2	−0.110624	−0.040689	−0.176777	
±3	−0.023849	−0.064539		
±4	0.037828			

2.2.3.1 DWT Decomposition Operations

The DWT decomposition operation consists of two steps namely; filtering and decimation (down sampling by 2). This operation is reversed at the decoder side in the reconstruction operations. It acts to reverse the decomposition operations using inverse wavelet filters and up sampling by 2. The filtering operation is accomplished by applying a sequence of low and high pass filters. The DWT decomposition and down sampling operation for a 1-D input signal can be expressed by the following two equations [77]:

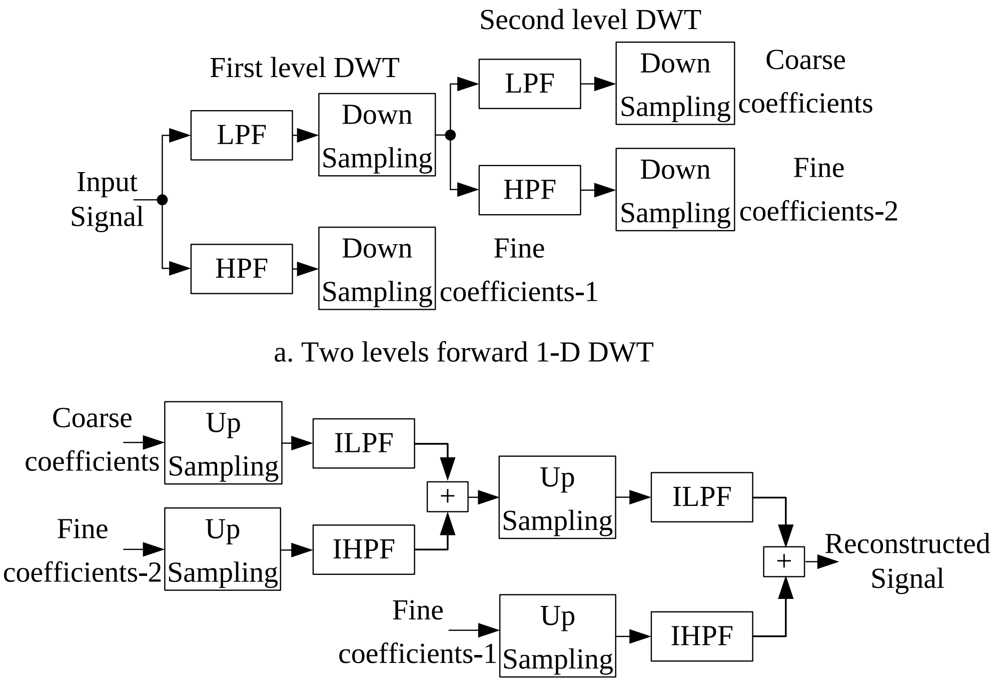
$$\begin{aligned} Y_{Low}(n) &= \sum_{k=-\infty}^{\infty} g(k-2n)\,x(k) \\ Y_{High}(n) &= \sum_{k=-\infty}^{\infty} h(k-2n)\,x(k) \end{aligned} \tag{2.9}$$

Where $x(k)$ represents the input signal and $Y_{Low}(n)$ and $Y_{High}(n)$ are the low and high output bands, and g and h represent the impulse response of DWT low and high pass filters, respectively. Further, the reconstruction operations can be carried out as follows [77]:

$$x(k) = \sum_{n=-\infty}^{\infty} g'(k-2n)\, Y_{Low}(n) + \sum_{n=-\infty}^{\infty} h'(k-2n) Y_{High}(n) \qquad (2.10)$$

where g' and h' represent the impulse response of the inverse DWT (IDWT) low and high pass filters, respectively.

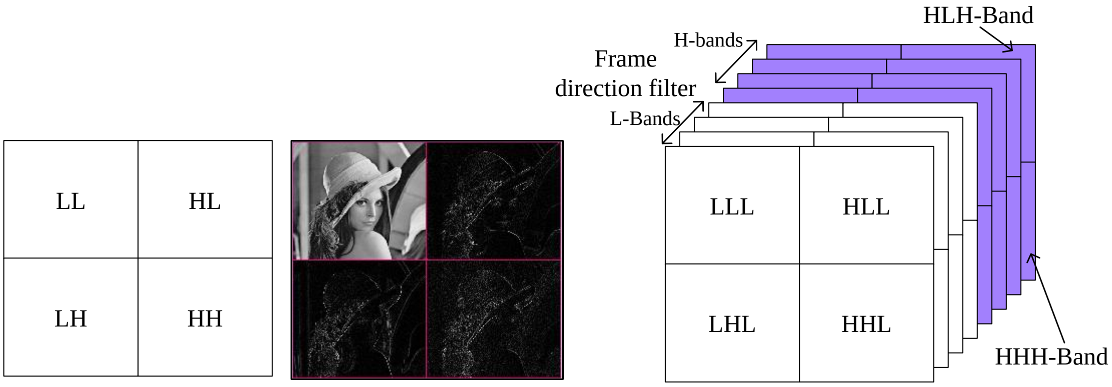
The decomposition and reconstruction proposes can be repeated L times, where $L = log_2 P$, P is the input data length and L is the number of decomposition levels, which leads to a multilevel 1-D DWT decomposition and reconstruction as shown in Figure 2.3. This figure shows that both directions (decomposition and reconstruction) consist of a set of low and high pass filters with down sampling step in the decomposition and an up sampling step for reconstruction. Furthermore, in Figure 2.3 the upper output represents the coarse components while the fine components of the original signal are represented in the lower part. The coarse part contains the shape of the signal or image decimated by the number of wavelet levels. It results from applying a sequence of low pass filters only. The fine parts are the output that results from a combination of high and low pass filters or high pass filters only. Furthermore, in multidimensional signals such as 2-D images and video sequences, the DWT can be computed using separable technique by applying 1-D DWT on each dimension, successively. Thus, the 2-D DWT can be computed by applying (2.9) on the rows and columns and on frames in the case of 3-D DWT. Further , 2-D and 3-D DWT can be computed using non-separable technique by applying 2-D and 3-D wavelet filters on the input data. Therefore, for the 2-D DWT, four output bands are computed for each decomposition level, namely: LL, HL, LH and HH bands, where: L and H stand for 1-D Low and High pass filters, respectively. Similarly, the output arising from 3-D DWT is 8-band signal. These bands are: LLL, HLL, LHL, HHL, LLH, HLH, LHH and HHH. As a result of the filtering operations, H bands contain the fine components and the L band contains the coarse components in 2-D and 3-D DWT. The output bands for one level 2-D and 3-D DWT decomposition are shown in Figure 2.4.



a. Two levels forward 1-D DWT

b. Two levels inverse 1-D DWT

Figure 2.3: Typical two levels 1-D DWT: a. Decomposition b. Reconstruction



(a) 2-D DWT decomposition

(b) 3-D DWT decomposition

Figure 2.4: One level 2-D and 3-D DWT decomposition.

The convention-based 1-D DWT computation of any 1-D signal can be obtained by two convolution steps using low and high pass filters and down sampling by 2. The convolution operation produces two sub-bands, low and high frequency; each one contains half the number of the original input samples [78]. Thus, the DWT decomposition and down sampling operation for a 1-D input signal is computed using (2.9) and the reconstruction operations can be performed using (2.10) [77, 79].

The preceding computation procedure produces wavelet coefficients for 1-D signals, and moreover, the wavelet coefficients of 2-D and 3-D signals can be computed by applying 1-D DWT along each signal dimension, separable approach [78]. Additionally, as the convolution-based DWT computation approaches require high computation loads, a new algorithm was introduced in [80], which has been named the lifting scheme or lifting-based DWT computation method. The lifting scheme has been considered as an efficient approach for DWT computation. The computation load of the convolution-based DWT has been reduced by up to 50% using the lifting-based DWT scheme [81].

The lifting scheme partitions the wavelet filter into a sequence of triangular matrices. Such an approach is called lifting-based DWT. The lifting scheme is composed of a sequence of prediction and updating steps as shown in Figure 2.5. The most important features of this scheme can be summarised as follows [81]:

1. Faster than the ordinary convolution-DWT approaches.
2. Low memory requirement.
3. The computation of each lifting step can be implemented in parallel.

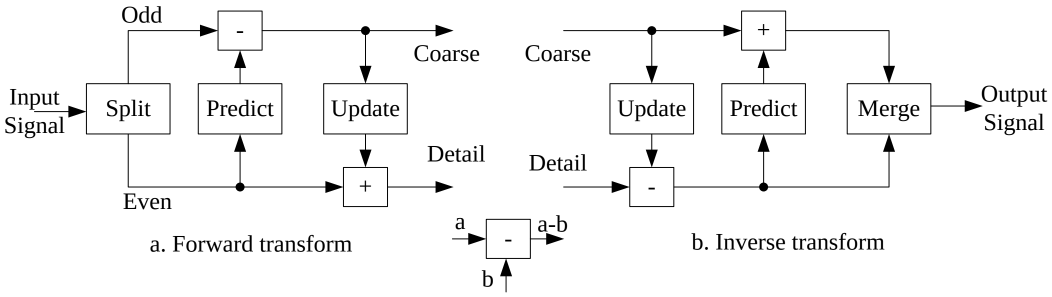


Figure 2.5: Lifting scheme (a) decomposition (b) reconstruction.

Further, the computational complexity of lifting-based CDF 9/7 DWT is shown in Table 2.3. The number of operations is computed according to the original structure of the CDF 9/7 1-D DWT lifting scheme. In this table, $3N$ multiplication and $4N - 4$ addition operations are required to compute the 1-D CDF 9/7 wavelet filter for N-sample using lifting scheme. In addition, 2N 1-D DWT is required to compute the 2-D

DWT for $N \times N$-point data block using the RC approach while, $3\mathrm{N}^2$ 1-D DWT is required in the case of the RCF 3-D DWT.

Table 2.3: The arithmetic operations of the lifting-based 1-D, 2-D and 3-D CDF 9/7 wavelet filter.

Operations	1-D	2-D	3-D
Multiplication	$3N$	$6N^2$	$9N^3$
Addition	$4N-4$	$2N(4N-4)$	$3N^2(4N-4)$

2.2.3.2 DWT Applications

The JPEG2000 image compression standard is one of the most important applications of 2-D DWT [7, 82]. Further, the multidimensional DWT is used in hyperspectral image compression [32], 3-D integral images compression [83], medical image compression [26, 84-86] and image watermarking [87]. The 3-D DWT is used in video, medical and remote sensing imaging (multispectral images) compression systems, and such an example is the 3-D SPECK [32]. Furthermore, various other wavelet-based image compression algorithms have been published in the literature. Among them the Embedded Zerotree Wavelet (EZW), Set Partitioning In Hierarchical Trees (SPIHT), Set Partitioned Embedded BloCK coder (SPECK), Embedded Block Coding with Optimized Truncation (EBCOT)[32].

2.3 Rate Distortion Measurements

Image and video quality is an important factor in different applications. Thus, the output accuracy through this thesis is computed and considered carefully using the well-known rate distortion measurement criteria. The Root Mean Square Error (RMSE) and Peak Signal to Noise Ratio (PSNR) have been used as objective fidelity criteria. The RMSE and PSNR (dB) between the original and the reconstructed images or frames can be computed as follows:

$$RMSE = \sqrt{\frac{1}{PQ}\sum_{i=1}^{Q}\sum_{j=1}^{P}\left(I_{in}(i,j) - I_{out}(i,j)\right)^2} \tag{2.11}$$

$$RMSE = \sqrt{\frac{1}{PQF}\sum_{i=1}^{Q}\sum_{j=1}^{P}\sum_{k=1}^{F}\left(I_{in}(i,j,k) - I_{out}(i,j,k)\right)^2} \tag{2.12}$$

$$PSNR = 10log_{10}\left(\frac{\left(max(I_{in})\right)^2}{(RMSE)^2}\right) \tag{2.13}$$

where, P and Q are the image/frame dimensions, F is the number of images or frames, $I_{in}(i,j)$and $I_{out}(i,j)$ is the original and reconstructed images, respectively. $Max(I_{in})$ is typically equal to 255.

2.4 Related work

The related work section has been divided into three sub-sections to summarise the previous work about the AQC, DCT and DWT algorithms and architectures.

2.4.1 AQC Related Work

Image compression can be classified into lossy and lossless classes. The lossy class reduces the bits required for storing or transmitting an image without much consideration of the image resolution. Lossless compression preserves the quality of the compressed image so that it is exactly the same as the original.

Many algorithms have been developed by researchers for both image compression classes; some of these algorithms use the DCT or DWT, while others avoid the complexity of applying such transforms. Examples of transform-based algorithms are the JPEG and JPEG2000. The two standards are DCT and DWT based image compression standards, respectively. The main issues with JPEG and JPEG2000 are their unsuitability for small devices that require low power consumption and high processing speed.

The other class of compression algorithms avoids using the DCT or DWT. A typical example of such algorithms is the BTC, which was introduced by Delp and Mitchell [2]. This algorithm is used for image compression with a bit rate of 2 bit per pixel (bpp) and leads to good image quality. The algorithm divides the input image into 4×4-pixel blocks to compute a bit plane using a two-level quantiser, and two moments representing the high and low mean. The main advantages of BTC are the good image quality and low computational complexity, compared to the transform-based compression algorithms. However, the main drawback of this algorithm is the annoying blocking artefact caused by the low bit rate requirement [1, 3, 37].

Many researchers have tried to modify or enhance the performance and reduce the bit rate of the BTC [4, 5, 37, 38, 40-42, 88]. Some of these algorithms utilise the same BTC two-level quantiser, while others use a three-level quantiser. In [37], a three level quantiser has been used to classify the pixels into low, intermediate and high intensity, according to predefined thresholds. To further reduce the bit rate, this algorithm transmits only odd rows and columns of each 6×8-pixel block; the remainder of the pixels are reconstructed according to an interpolation function. In [4], an algorithm for colour image compression was presented as a combination of the ordinary BTC and the AQC [5]. This algorithm converts the RGB image to a Luminance-Chrominance colour space (YC_bC_r) and partitions it into 6×6-pixel main blocks, and each block is divided into four 3×3 sub-blocks. The four 3×3-pixel sub-blocks are encoded using the BTC algorithm to extract four luminance bit planes and 24 moments (two moments for each sub-block in each colour plane). These components are encoded using the AQC algorithm. Further, an improved BTC algorithm for colour image compression was presented in [42]. This algorithm generates a luminance bit map to represent the edge information, with three additional bits to represent the differences between luminance (Y) and other colour planes. Furthermore, the BTC blocking artefact is reduced using modified Error Diffusion (EDF) by diffusing the quantised error into neighbouring pixels [40]. It utilises three EDF kernels to distribute the error between the original and the corresponding bi-level BTC pixels into the neighbourhood pixels. Further, an improved and low complexity BTC algorithm was presented using a look up table (LUT) dither array [41]. In this algorithm, the dither array is calculated offline to reduce the computation complexity. The reduction in blocking artefact is an important feature of this algorithm. An improved multitone BTC algorithm was presented in [88]. This algorithm uses a 3×3 mean filter at the decoding phase to produce an intermediate image. The intermediate image is used to compute a bilateral filter weights. Then, the multitone image is computed by applying the bilateral filter to the decoded image.

2.4.2 DCT Related Work

Transforms such as the Discrete Fourier Transform (DFT), DWT, and DCT play a critical part in various DSP applications, including audio, image and video systems. Much of the usefulness of these transforms arises from their frequency and time-frequency representations and properties including the decorrelation property, energy compactness and the availability of fast algorithms for their computation. Nevertheless, even the fast algorithms that implement these transforms are very compute-intensive. Thus, these transforms can become a bottleneck in terms of the speed of the system, and contribute greatly to the area usage and power consumption of their wide applications [10-17, 22, 27, 28, 89-92]. For its role in many image and video applications, including the JPEG, MPEGx and H.26x compression standards, the DCT has received a great deal of research interest [6, 8, 51-53]. As a matter of fact, the 2-D DCT is the image transform which reduces inter-pixel redundancy in the JPEG standard [6]. Further, 3-D DCT is used to eliminate the need for motion estimation in video compression systems, which takes advantage of the data redundancy of video stream in spatial and temporal domains [54]. Numerous 1-D and 2-D DCT architectures have been suggested in the literature [93-99]. Exploiting the separability principle of the transform, 2-D DCT cores are based on the 1-D DCT RC approach [95-97, 99]. In addition, they address different requirements, including power consumption, speed and hardware usage [10-17]. While the 3-D DCT architectures can be found in [24, 25, 100-103]. Traditionally, the 3-D DCT has been implemented by cascading stages of the 1-D DCT, taking account of the separability principle which breaks the 3-D DCT computation to three 1-D DCTs, such as in the well-known RCF approach. Noteworthy differences between architectures in the literature is their level of parallelisation in terms of the number of stages and the number 1-DCT units per stage, which leads to different trade-offs between circuit complexity and throughput rate. One common architecture employs three stages of one 1-D DCT unit each and $N^3 + N^2$ transpose memory [24, 102, 103]. Parallelisation can be applied to the first two 1-D DCT processors, which indeed implement a 2-D DCT transform, leading to the utilisation of $2N+1$ 1-D DCT processors and N^3+N memory. Another class of the 3-D DCT architectures multiplexes the 1-D DCT involved in its computation into single 1-D DCT architecture. Such a class of architecture requires N^3 memory words [24, 103]. In [24], a context based 3-D DCT RCF algorithm and its VLSI implementation for various parallelization levels has been introduced. With regards to 180 nm CMOS technology, such architecture can operate at frequencies of up to 72, 103 and 120 MHz for 1.6, 1.8 and 1.95 supply voltages, respectively. Furthermore, for 90 nm, the proposed system operates at frequencies of up to 120.2, 176.4 and 208.8 MHz using 1.08, 1.2 and 1.26 supply voltages, respectively [24].

Evidently, the economy made in hardware utilisation came at the cost in throughput rate. As a matter of fact, using three 1-DCT architectures to implement the 3-D DCT achieves a throughput rate three times higher than when employing a single 1-D DCT processor. Another class of the 3-D DCT architecture relies solely on the systolic approach with its well established methodology and complexity analysis [25]. In [25], a 3-D DCT architectures based on a dedicated three-dimensional array processor have been presented. Such architectures are termed; Original, Pipelined ver.1, Pipelined ver.2 and Block. These architectures utilises RCF technique and different levels of parallelisation and pipelining. Nevertheless, such parallelisation levels are associated with high hardware utilisation costs. As an example, for $N = 8$ points the block architecture consumes 19,355 Logic Elements (LE), 13,347 registers, 525 input/output (I/O) pins and 128 embedded multipliers.

2.4.3 DWT Related Work

The DWT is an efficient tool in multidimensional signal processing and digital communication applications. While the 1-D DWT has found in many audio applications, the 2-D DWT is used in the well-known image compression JPEG2000 and MPEG-4 standards, watermarking, de-noising and multimedia information systems [7, 18, 78]. The lossless JPEG2000 compression system utilises the lossless 5/3 integer wavelet filter, termed Daub-4, while the CDF 9/7 floating-point filter banks are used in the JPEG2000 lossy compression system [104]. Furthermore, the 3-D DWT is used in video, medical, hyper spectral and integral image compression systems [28, 83, 84, 105]. Although the DWT has more applications than the DCT, it is affected by high computational complexity and high memory requirements [28]. Therefore, numerous algorithms and architectures have been proposed in the literature for efficient DWT implementations with the aim of lower-power consumption, higher clock rates and lower memory requirements [18-23, 27-30, 86, 105-110].

A VLSI 2-D DWT architecture for lifting-based DWT computation is proposed in [18] .Such an architecture is based on a modified lifting scheme for CDF 9/7 1-D and 2-D wavelet filters. In [19], 2-D lifting based DWT multi-input/multi-output VLSI architectures were proposed. The proposed architectures process multiple input data rows concurrently to attain high processing speeds. The operating frequencies of the proposed architectures were 65.38, 64.25 and 63.52 MHz for 2, 4 and 8 parallel computation units, respectively, with a memory requirement of $10M+4N+MN/2$ where M is the number of parallel units and $N\times N$ is the image size. Furthermore, in [20], parallel FIR filters with a polyphase structure were used to improve the speed of the 2-D

DWT structures. A high speed non-separable pipeline 2-D DWT VLSI architecture was proposed in [21]. Such architecture can operate at a frequency of up to 135 MHz with 8×18 Kb block RAM. In [22], high-throughput and low-latency 2-D block-based pipeline DWT architectures using lifting scheme were proposed. In addition, a high-throughput lifting-based 2-D DWT architecture for efficient area and power consumption was proposed in [23]. The memory requirement of the proposed architecture is ($4N$ + $8P$) words, where N is the image width and P is the input block size.

Furthermore, in [28], a lifting-based 3-D DWT architecture was proposed for an infinite Group Of Frames (GOF) input data. This architecture can operate at a frequency of 321 MHz and generates 2 samples /clock cycle. A VLSI architecture which implements the Daub-4 using the conventional convolution based method was proposed in [29]. The memory requirement of such architecture is $N^2/4 + 4N$, where $N \times N$ −pixel is the frame size. Furthermore, the proposed architecture in [109] uses on-chip and off-chip memories of $2F(K-2)(N+2) + 8K$ and $FN^2/8$, respectively, where, F is the number frames, K is the filter length and $N \times N$ is the frame size. The on-chip memory represents the internal block memory slices while the off-chip memory represents the external memory resources such as the DDR. In [30], cascaded pipeline architectures for multilevel convolution-based 3-D DWT computation were proposed. Such architectures use computation blocks to compute each decomposition level to reduce the frame-memory and maximize the hardware utilisation efficiency. Each decomposition level is split into three parallel stages to further improve the processing speed. However, the memory requirement of the proposed architecture is $2(K-1)Nx_2 + 2(2K-1)MN\frac{x_3}{3}$ on-chip storage and $\frac{MNx_4}{6}$ frame buffer, where, M: image height, N: image width, K: is the filter length, $x_2 = 1 - 2^{-J}$, $x_3 = 1 - 2^{-2J}$, $x_4 = 1 - 2^{-2J+2}$ and J is the maximum of levels of 3-D DWT decomposition [30]. A memory-efficient 3-D DWT architecture using an overlapped GOF has been proposed in [27]. In this architecture, the input frames are partitioned into blocks of 6×6-pixel, while the GOF and the number of overlapped frames are related to the wavelet filter order. As an example, the GOF of the Daub-4 is equal to 4 and two frames have to be overlapped. Similarly, the GOF of the CDF 9/7 is equal to 9 frames with 7 overlapped frames. In general, the number of overlapped frames is equal to GOF-2 while the GOF is equal to the filter order. Thus, for the Daub-4, four frames are fed to the computation units, and two of them have to be reloaded again in the subsequent GOF. Therefore 50% data redundancy is required.

Moreover, in the CDF 9/7 wavelet filter, 7 frames should be reloaded to the architecture together with two new frames to provide sufficient data for 3-D DWT computation.

2.5 Field Programmable Gate Array (FPGA)

The FPGA is a type of Application Specific Integrated Circuits (ASIC) which is designed to be configured by the hardware designer using different hardware description languages [111]. An example of FPGA device suppliers is the Xilinx Corporation, which has been one of the leading FPGA devices providers since 1984. Xilinx have introduced relentless variant FPGA devices with system performance, processing power, hardware resources and power consumption. Up to 2010, two main Xilinx programmable families had been introduced, *Virtex* and *Spartan*. They are largely similar in terms of construction and architecture. However, the Virtex family can be used for high performance applications. It is fabricated in more advanced CMOS technology, while the Spartan family is more applicable for low-cost, high volume applications [112]. A brief description of the Xilinx FPGA families that have been introduced since 2003 is illustrated in Table A.1 in the Appendix A, taking the main characteristics of the largest device from each family for the sake of a clear comparison. A quick view of Table A.1 reveals that the block memory sizes have increased up to 10 times and so has the number of Digital Signal Processing (DSP) slices. For the DSP slices, it is clear that the earlier versions of Xilinx devices was only composed of 18×18 bit multipliers, while in the most recent devices the DSP slices have become more advanced and contain 25 × 18 bit multipliers, together with an adder and accumulator. Another difference lies in the Configurable Logic Block (CLB) construction. CLBs contain two slices, each slice has four 6-input Look Up Tables (LUT) and eight Flip Flops (FFs) in the recently introduced Xilinx device, while they used to have four slices per CLB and each slice used to have two 4-input LUT and two FFs.

Further, in 2010 another two families were introduced, Artix 7 and Kintex7, together with a high performance Virtex 7 family. A comparison between these families and series 6 families can be found in Table A.2 in the Appendix A [113]. It is obvious from Table A.2 that Virtex 7 is more efficient than Virtex 6 by approximately 50% in terms of logic cells, Block RAM, DSP slices and power consumption. Furthermore, the Xilinx series 7 represents the lowest power consumption family among all other Xilinx families; in comparison with the previous family, series 7 has 65%, 25% to 30% and 30% lower static, dynamic and I/O dynamic power consumption, respectively [114].

Furthermore, the series 7 Virtex 7 family is a high performance family in terms of performance while Artix 7 is suitable for low power consumption applications. A simple comparison between Virtex7, Kintex7 and Artix7 in terms of performance and power consumption is illustrated in Figure A.1 in the Appendix A [114].

Furthermore, the most recent Xilinx product is the Zynq 7000 automotive family, which is fabricated using 28 nm CMOS technology based Xilinx programmable logic. This family mixes all the programmable 28 nm technology with a dual-core ARM Cortex A9 processor, as described in Figure 2.6. The ARM processor is a 64-bit microprocessor which is used in many mobile phones and other small devices. The ARM processing system comprises on-chip memory and external memory interfaces together with the ARM Cortex-A9 processor [115, 116].

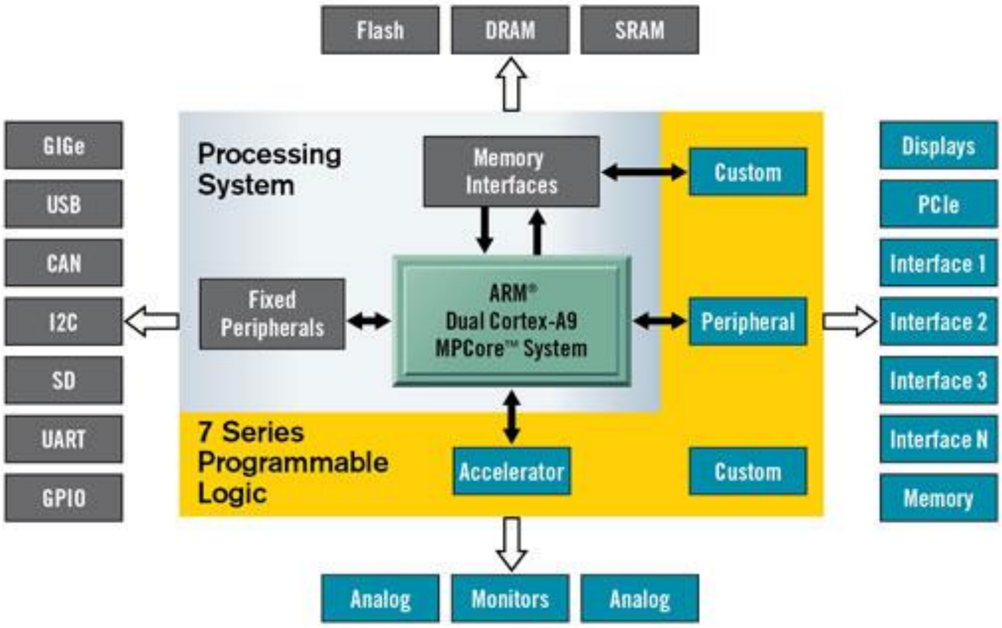


Figure 2.6: Zynq-7000 system architecture [117].

2.6 Xilinx FPGA Devices' Main Components

Xilinx FPGA devices are a composition of CLBs, DSP blocks, Block RAMs, Input/output ports (I/O) and programmable interconnections, as shown in the basic block diagram of Figure 2.7 [118-125].

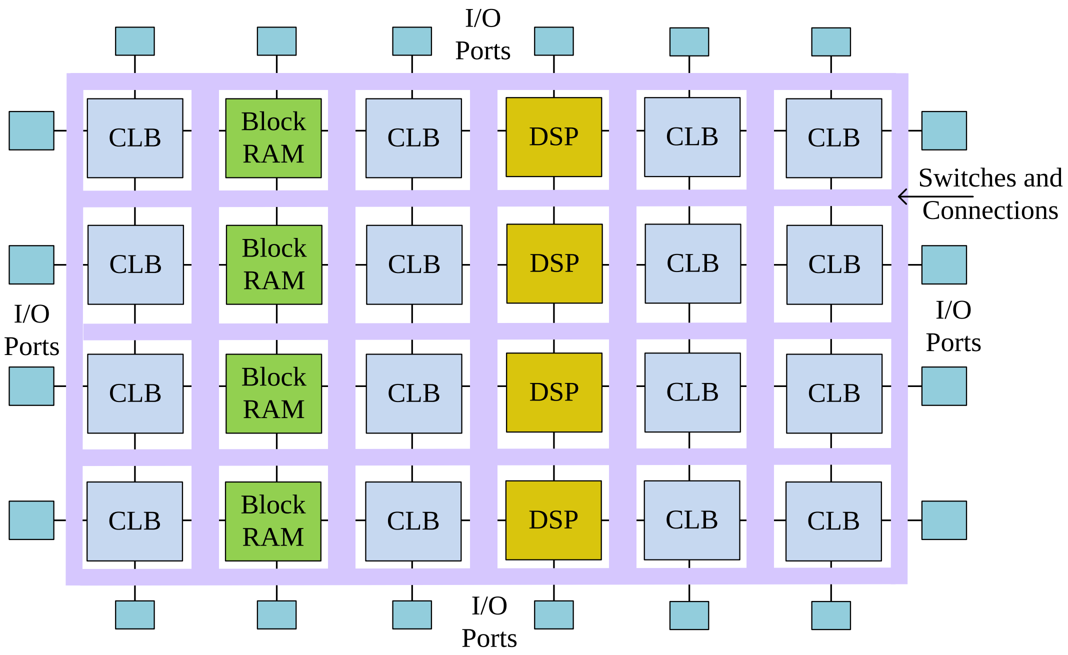


Figure 2.7: Basic block diagram of an FPGA device architecture [126].

2.6.1 *Configurable Logic Blocks (CLBs)*

A CLB is the main logic component used for circuits implementations. It composes of two slices (four slices in Virtex 4, see Table A.1 in the Appendix A) connected to a switch matrix, as shown in Figure 2.8. Each slice has its own carry chain [120, 121]. Two types of slices are used in FPGA devices, namely: SLICEL and SLICEM. SLICEL is a combination of LUTs, storage elements, wide function multiplexers and carry logic. It can be used to implement logic and ROM functions. The function generators can be configured as 6-input LUTs or dual-output 5-input LUTs. The Boolean function of each LUT can be modified or altered using existing hardware description languages. SLICEM has an additional function which supports data shifting using 32-bit registers (or 16-bit × 2 shift registers) and data storing using distributed RAM (64-bit distributed RAM). A brief description of CLB logic resources for both slice types is shown in Table 2.4. It is clear from Table 2.4 that Virtex 6 has twice the number of Flip-Flops of Virtex 5; as such it provides more efficient hardware functionality.

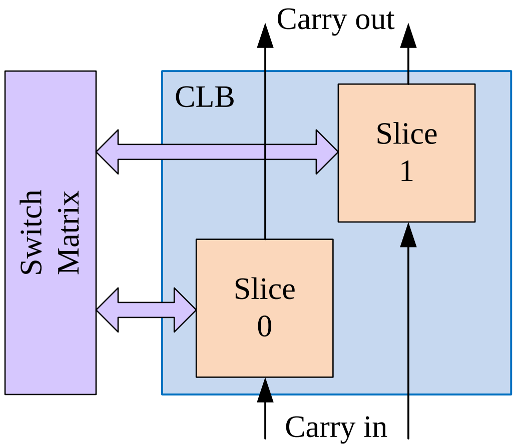


Figure 2.8: CLB block diagram [120].

Table 2.4: Hardware resources in single CLB for Virtex 5 and Virtex 6 [120, 121].

Device	All CLBs (SLICEL and SLICEM)				SLICEM only	
	Slices	LUTs	FFs	Arithmetic and Carry Chains	Distributed RAM (bit)	Shift Registers(bit)
Virtex 5	2	8	8	2	256	128
Virtex 6	2	8	16	2	256	128

2.6.2 *Block RAMs*

The block RAM is a dedicated hardware resource which can be used to store large data for high speed applications. Up to 18, 38 and 68 Mb block memory can be constructed using available block RAM resources in Virtex 5, 6 and 7 families, respectively. Further, each block RAM can be constructed as two independently controlled 18 Kb RAMs or one 36 Kb RAM. In addition, each 36 Kb RAM can be cascaded with other 36 Kb RAMs to form bigger block memories. A variety of memory components can be implemented using a Xilinx CORE generator [124]. This includes ROMs, single/dual port RAMs and FIFOs. Since the FIFO uses dedicated memory resources, it does not require additional logic resources for its data control unit [120-122].

2.6.3 Distributed RAM (Available in SLICEM only)

Distributed RAM is an efficient memory component in terms of resources, performance, and power aspects. It represents a good solution for small data and it can be implemented using available SLICEM LUTs resources [120]. Generally, Distributed RAM should be used for data sizes of 64 bits or less. However for a depth varying from 64 to128 bits, the decision of using block or distributed RAM depends on the availability of block RAMs; if not available, distributed RAM can be used instead. Further, the block memory should be used for data widths greater than 16 bits [127].

2.6.4 DSP Slices

DSP slices accelerate and reduce the power consumption required by the multiplication operations. Each DSP slice in Virtex 5 and later families composed of 25×18-bit two's complement multiplier with a 48-bit output precision and optional adder, subtracter and accumulator with an optional pipeline stages for performance enhancement [118, 123, 128].

Further, each Xilinx device contains interconnection matrices, I/O buffers and ports, PCI interfaces, high throughput transceivers and memory interfaces. Full details about the components of each Xilinx FPGA device can be found in [125].

2.6.5 Specifications of FPGA Devices Used in This Thesis

In this thesis, two Xilinx FPGA devices are used for verification and performance evaluation. The first device is the xc5vlx50t-3ff1136 Virtex 5 FPGA. It is used in the evaluation operations of the architectures in chapters 3, 4 and 5. Such a device encompasses up to a 2.1 Mb block memory and 48 DSP slices; both of these are required for the proposed architectures. The second device is the XC6VLX760 Virtex 6 FPGA device. It is used in Chapter 6 to fulfil the hardware requirement of the parallel multidimensional wavelet architectures. The main features of this device include; a block memory of up to a 25Mb and 864 DSP slices. A summary of the main features of the two FPGA devices is shown in Table 2.5 [118-121].

Table 2.5: Summary of XC5VLX50T and XC6VLX760 Xilinx FPGA devices characteristics.

Device		Virtex 5 XC5VLX50T	Virtex 6 XC6VLX760
CLBs	Total Slices	7200	118560
	SLICELs	5280	85440
	SLICEMs	1920	33120
	Max Distributed RAM (Kb)	480	8280
	Shift registers (Kb)	240	4140
DSP48E1 Slices		48	864
Block RAMs	18 Kb	120	1440
	36 Kb	60	720
	Max (Kb)	2160	25920
Max User I/O		480	1200
Slice configuration		4-LUT, 4-Storage elements (FF), multiplexers and carry logic	4-LUT, 8-storage elements (FF), multiplexers and carry logic

2.7 Xilinx FPGA Design Tools

Various Xilinx FPGA designing tools can be used in FPGA devices programming. They include an Integrated Software Environment (ISE) Design Suite, PlanAhead, an Embedded Development Kit (EDK), ChipScope Pro, AccelDSP, System Generator for DSP and Vivado High Level Synthesis design suite (Vivado HLS) [125].

In ISE design suite and PlanAhead, the designed system can be built schematically or by programming using VHDL and Verilog hardware description languages. Through the ISE design suite, the designer can synthesise, implement, verify and perform device configuration for the required design throughout the project navigator’s panes. The EDK tool includes Xilinx Device Studio, the Software Development Kit, documentation and IPs for designing embedded processing systems using Xilinx FPGA Devices. Moreover, the ChipScope Pro tool captures, display and analyse the internal signals for a selected node. Another hardware design tool is the AccelDSP Synthesis tool [129]. It is used to transform Matlab floating point design into a fixed-point either in Matlab or

C++ format, and then it can be implemented in a Xilinx FPGA device. Further details can be found in [125].

Moreover, two other high level hardware design tools can be used in hardware architectures implementations. These two tools include; the Xilinx System Generator for DSP and Vivado HLS.

2.7.1 Xilinx System Generator for DSP

System Generator for DSP is a high level hardware design tool [130]. It is used for high performance DSP system implementation using Xilinx FPGA devices. The Xilinx system generator is a system-level modeling tool based on Simulink of Matlab [125]. It shortens the time required to design and implement efficient DSP algorithms compared with other traditional Register Transfer Level (RTL) design procedures. It merges the functionality of Matlab Simulink with the requirement of hardware implementations using new Simulink DSP blockset that have been added to fulfill the hardware requirements. The DSP blockset is a library containing over 90 DSP blocks such as adders, multipliers, block RAMS, multiplexers, and others, as shown in Figure 2.9. The DSP blocks can be connected together in an appropriate way to satisfy the requirement of any proposed DSP system. Furthermore, the Xilinx System Generator for DSP generates VHDL or Verilog codes automatically from Simulink workspace, while the VHDL codes can be imported using a specific block in order to use it with other existing blocks as well. Further, a sequence of optimization operations should be carried out in order to improve the performance and robustness of the proposed architecture. The optimization operations include validating the functionality of the proposed architecture, whether it meets the design constraints or not. The function of the proposed architecture can be verified using Matlab Simulink simulation workspace with an appropriate setting of simulation time and clock period. Moreover, the power consumption can be analysed using Xilinx Power Analyzer, which is an integrated tool with the System Generator. Also, the timing constraint and critical path delay can be viewed and optimized for better performance and higher processing speed. Further, the critical path delay can be reduced by inserting some pipeline stages as required. The main System Generator DSP blocks shown in Figure 2.9 are the following [130]:

- System Generator block: It is used to set the system clock period, select the target hardware device and VHDL and Verilog code generation. Further, another important function of System Generator block is the timing and power analysis generation. Through the timing and power analysis tool, the designer can ensure that the proposed system meets the timing and power constraints before it is embedded or merged with other systems in the large model.

- Gateway in: It represents the input port to convert the data from floating point to fixed point representation in a FPGA environment according to the word length and data precision specified by the designer.
- Gateway Out: It is used to convert the data from fixed point to floating point representation and acts as an output port.
- WaveScope: It is an optional tool to view the output waveforms at a dedicated node in the architecture.
- Dual port RAM, single port RAM, ROM and FIFO: In RAM, ROM and FIFO blocks the depth, word size, latency and initial data (if any) of the selected RAM/ROM can be set. These memories can be constructed as a block or distributed RAM with a specific latency, as required. The block memory is constructed using the available dedicated memory resources.
- Register: It is a D-FF register with one sample latency.
- Delay: It is a chain of Shift Register LUT (SRL) followed by FF. It is used for data synchronising and pipelining purposes.
- Shift: It is used to perform binary right and left shift with different output precision levels.
- Slice or bit/bits slice: It is used to select specific bit/bits from the input sequence to create new data values with unsigned integer output values.
- CMult: It performs the multiplication of the input data by a pre-selected constant value. Further, its latency and the output precision can be adjusted for optimum and fast multiplications.
- Mult: It is used for two input data multiplication. Its latency and the output precision can also be adjusted.
- AddSub: It is an adder/subtractor with the ability to modify latency and output precision. It can be used as an adder or subtractor or both.

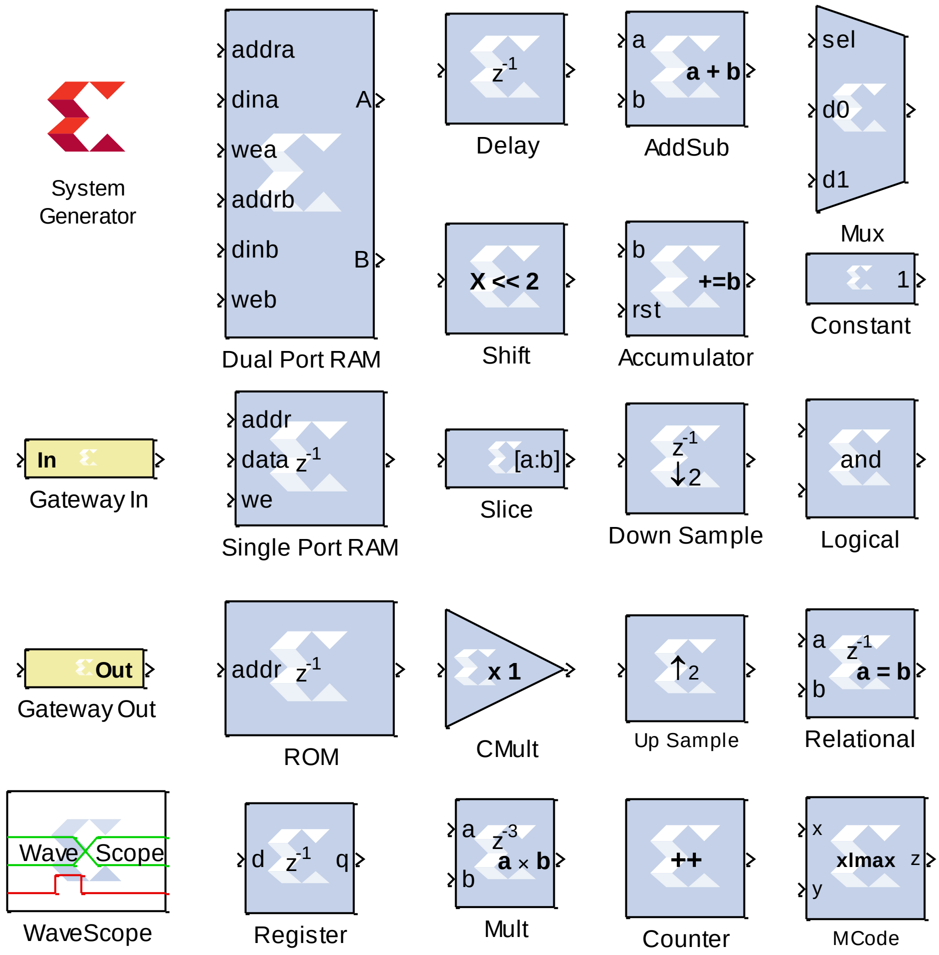


Figure 2.9: Selected DSP blocks from system generator blockset.

- Accumulator: It performs an accumulation operation (addition or subtraction) with different feedback scales and different latency depths. Its output data precision is similar to that of the input data.

- Down sample and up sample blocks: Different sampling rates can be obtained using these two blocks; the up sampling block either repeats the data or fills it by a sequence of zeros.

- Counter: Different ranges of up or down counters can be implemented using such a block and the output of the counter can be specified as a signed or unsigned.

- Mux: It is a multiplexer with variable number of inputs and one unsigned control signal.

- Constant: It is used to set specific constant values such as an integer, fractional or Boolean.
- Logical: It performs logical operations such as AND, NAND, OR, NOR, XOR and XNOR with adjustable number of inputs.
- Relational: It is a comparator to perform relational operations such as greater than, equal or not equal.
- MCode block: Using the MCode block, the designer can use simple Matlab code to perform simple operations such as comparison and/or switching.
- Black Box: This is used to import pre-designed VHDL or Verilog codes to be used with other existing DSP blocks in the Simulink blockset.

2.7.2 Vivado High Level Synthesis Design Suite (Vivado HLS)

The Vivado HLS is the most recent Xilinx design tool. It was unveiled in April 2012 to support faster hardware implementation and verification. The Vivado HLS accelerates IP generation by directing C, C++ and SystemC codes according to the requirements of the Xilinx FPGA devices. Furthermore, it offers faster implementation and verification with an important feature of about 35 % average power advantage and about 20% better LUT utilisation[125, 131].

2.8 Architectures Design Procedure

A design may go through variant implementation and verification steps, as shown in Figure 2.10. The figure, starting point is to analysis and apperception of the targeted algorithm. In the second step, such algorithm is coded in Matlab for testbench purposes. Thus, this is the base of the whole design procedure; the results obtained from this step are used for verification of the architecture. In the third step, the architecture is built using the block sets of the system generator, and the architecture output is collected and verified with that obtained from Matlab codes. Various optimization operations are carried out to further reduce the hardware cost, power consumption and increase the processing speed. Such operations include pipelining to shorten the critical path, varying wordlengths for output precision, and the replacement of high cost hardware blocks, such as dividers and multipliers, with equivalent low cost blocks. Finally, once all the optimization procedures have been carried out, the results are collected and a comparison with the software implementation results is made.

Further, throughout the rest of this thesis the algorithms and architectures are designed and tested using the following software and tools:

1. Machine specifications: Intel (R) Core (TM) 2 CPU 6600 @ 2.4GHz, 3.24 GB of RAM and 80 GB of Hard disk.

2. Operating system: Microsoft Windows XP professional Version 2002 Service Pack 3.

3. Software tool: Matlab version 2010a.

4. Hardware design tool: Xilinx System Generator for DSP 12.4 (2010) and ISE 12.4.

5. Xilinx Xpower Analyser is used for power and timing analysis.

6. The testing data composes of standard images, video sequences and medical images. In chapter 3, 2-D images and individual frames with block sizes of $N \times N$-pixel have been used. While 3-D volume data and a cube size of $N \times N \times N$-voxel is used in the evaluation process of the 3-D DCT and 3-D DWT architectures proposed in chapters 4 to 6.

The design procedure using Xilinx system generator starts from Matlab Simulink and end with the bit stream generation, as shown in Figure 2.11 [132].

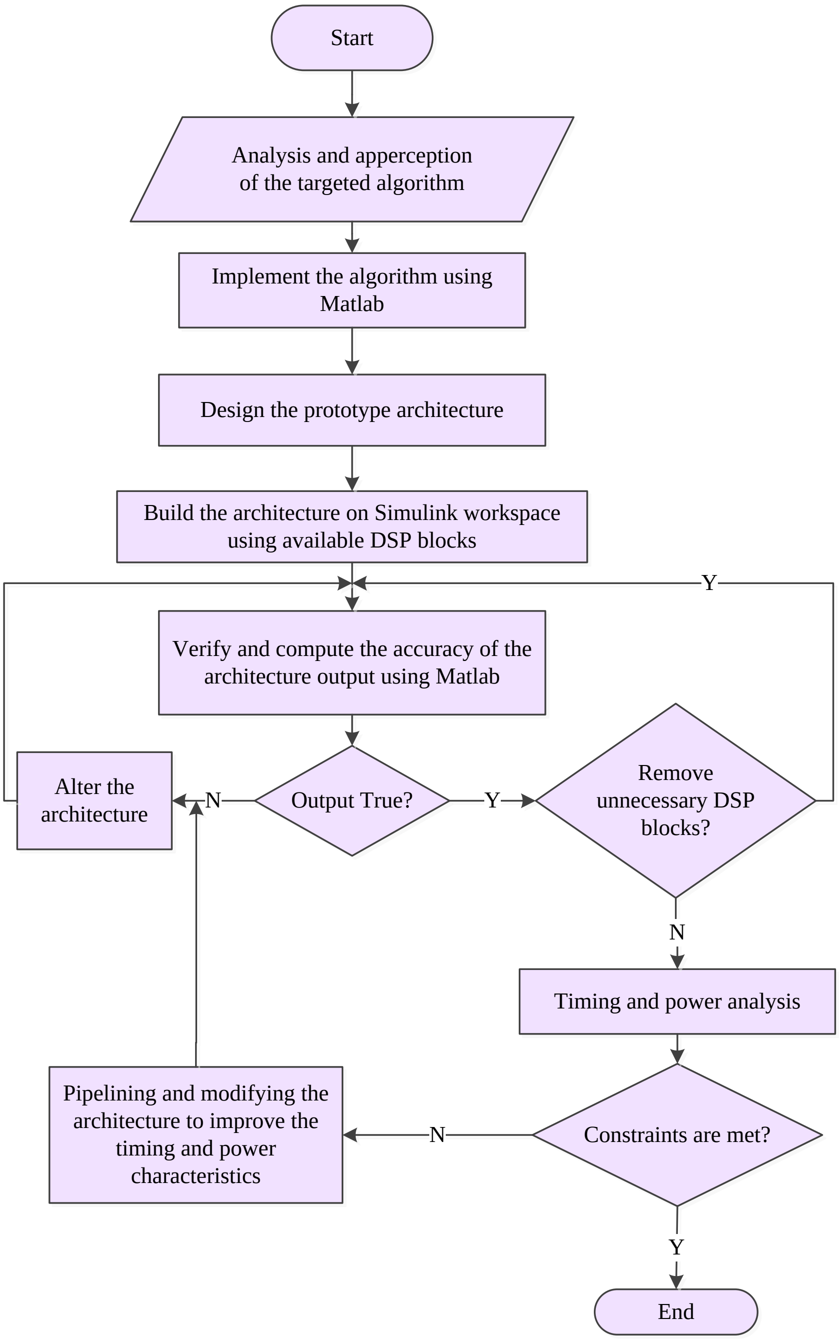

Start
Analysis and apperception of the targeted algorithm
Implement the algorithm using Matlab
Design the prototype architecture
Build the architecture on Simulink workspace using available DSP blocks
Verify and compute the accuracy of the architecture output using Matlab
Output True?
Alter the architecture
Remove unnecessary DSP blocks?
Timing and power analysis
Constraints are met?
Pipelining and modifying the architecture to improve the timing and power characteristics
End
Y
N
Y
N
N
Y

Figure 2.10: Architectures design procedure.

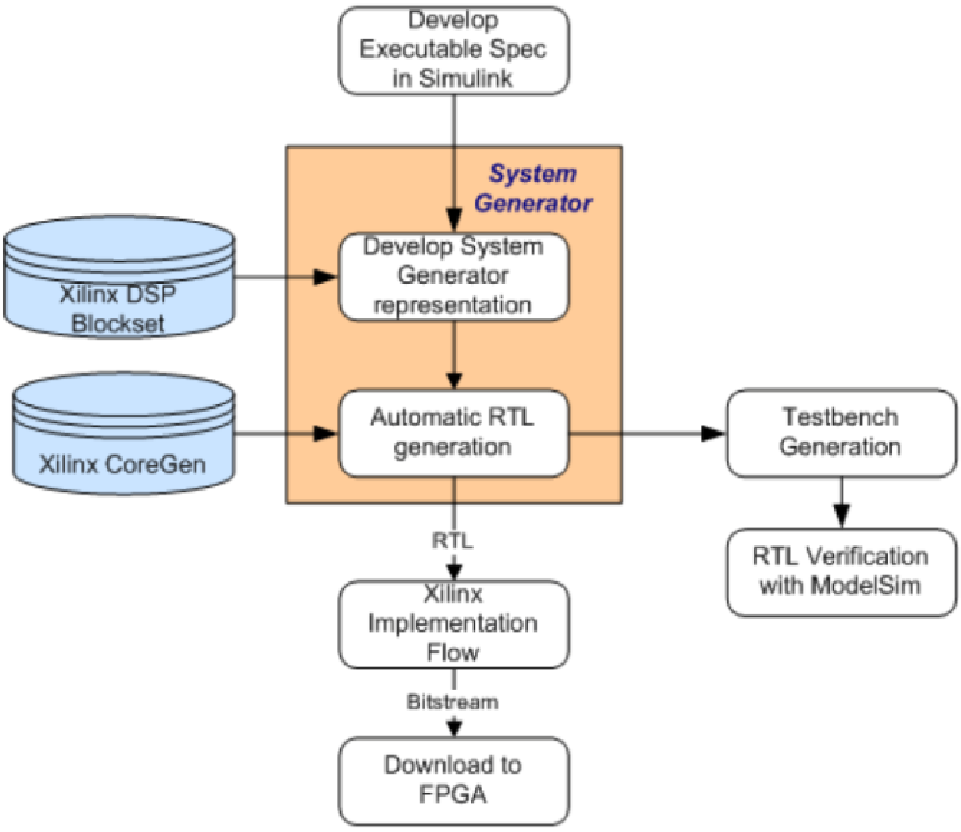

Develop Executable Spec in Simulink
System Generator
Xilinx DSP Blockset
Develop System Generator representation
Xilinx CoreGen
Automatic RTL generation
Testbench Generation
RTL
RTL Verification with ModelSim
Xilinx Implementation Flow
Bitstream
Download to FPGA

Figure 2.11: System Generator design flow [132].

2.9 Summary

This chapter has presented the information required to support the work in the rest of this thesis. The information provided has reviewed the low complexity image compression algorithms, the DCT and DWT, their computational complexity. Furthermore, as the rest of this thesis focuses on FPGA-based implementation of such algorithms using Xilinx devices, necessary key information about Xilinx FPGA families and their architectures has been given. Moreover, the related work and FPGA design tools were presented including Xilinx System Generator for DSP. Furthermore, the design procedure of the proposed architectures has been introduced and the design, implementation and verification steps are outlined.

Chapter 3: Low Complexity Block-Based Image Compression Systems

3.1 Introduction

Low complexity image compression algorithms are necessary for modern portable devices such as mobile phones and wireless sensor networks. Examples of such algorithms are the BTC and AQC. Hence, this chapter introduces two new low complexity block-based image compression algorithms for low bit rate applications and their high speed, low power consumption architectures. The proposed algorithms are based on the AQC algorithm by considering the intensity variation of image regions to reduce the bit rate of the encoded image. Thus, the proposed algorithms introduce the intensity check process as an additional parameter to the AQC parameters to control the output bit rate while preserving the output PSNR within an acceptable range. Such algorithms are termed Intensity Based Adaptive Quantisation Coding (IBAQC) algorithms. The key differences between the two proposed algorithms lie in the intensity check process of the image block and the computation cost. The first algorithm performs an intensity check operation on the variance of each image block prior to the AQC computation, whereas the second algorithm performs the same operation according to the quantisation steps of the AQC algorithm. The second algorithm avoids the computational complexity of the block variance required by the intensity check unit in the first algorithm. Moreover, high speed and low hardware usage architectures have been designed and verified using xc5vlx50t-3ff1136 Virtex 5 Xilinx FPGA device.

The remainder of this chapter is organised as follows: Section 3.2 introduces a background about the BTC and AQC algorithms. Section 3.3 presents two IBAQC algorithms, while section 3.4 presents their hardware architectures. The performance of both algorithms and their architectures are discussed in section 3.5 and the summary is drawn in section 3.6.

3.2 Background

3.2.1 Block Truncation Coding (BTC)

The BTC is a lossy image compression technique with a bit rate of 2 bpp [2]. The BTC algorithm uses a two-level quantiser and a bit plane which depends on local image features. The BTC algorithm partitions the input image into $N \times N$-pixel blocks, and computes the bi-level bit plane and two 8-bit moments for each block. The computation steps of this algorithm are as follows:

- Compute the mean and the standard deviation of each block.
- Specify a threshold so that each pixel within the 4×4-pixel block, which is smaller than the threshold, is encoded as *0*; otherwise, it is encoded as 1. In [2], the mean of each image block can be chosen as a threshold.

$$B_p(i,j)=\begin{cases} 0 & if\, B(i,j) < \bar{\mu} \\ 1 & otherwise \end{cases} \tag{3.1}$$

where $B_p(i,j)$ is the output bit plane, $B(i,j)$ is the intensity of each pixel in the block and $i,j = 1,2,..N$, and $\bar{\mu}$ is the block mean.

- Compute the two moments using the following two equations [2]:

$$L_0 = \bar{\mu} - \bar{\sigma}\ \sqrt{\frac{N_H}{N^2 - N_H}} \tag{3.2}$$

$$L_1 = \bar{\mu} + \bar{\sigma}\ \sqrt{\frac{N^2 - N_H}{N_H}} \tag{3.3}$$

where L_0 and L_1 are the moments, N_H is the number of pixels that are greater than the threshold of the block, and $\bar{\sigma}$ is the standard deviation of the block:

$$\bar{\sigma} = \sqrt{\overline{\mu^2} - \bar{\mu}^2}$$

$$\overline{\mu^2} = \frac{1}{N^2}\sum_{j=0}^{N-1}\sum_{i=0}^{N-1} B(i,j)^2 \tag{3.4}$$

$$\bar{\mu} = \frac{1}{N^2}\sum_{j=0}^{N-1}\sum_{i=0}^{N-1} B(i,j)$$

Thus, the encoding of each 4×4-pixel block requires 32 bits; 16 bits to encode the two moments and the remaining 16 bits for the resultant bit plane.

The computation load of the original BTC algorithm was reduced by the Absolute Moment Block Truncation Coding (AMBTC) [133]. The AMBTC algorithm replaces the standard deviation and the mean by an appropriate high and low means, as follows:

$$M_L = \frac{\sum\sum B(i,j)\ ,\ \{B(i,j)\leq\bar{\mu}\}}{N_L} \tag{3.5}$$

$$M_H = \frac{\sum\sum B(i,j)\ ,\ \{\ B(i,j)>\bar{\mu}\}}{N_H} \tag{3.6}$$

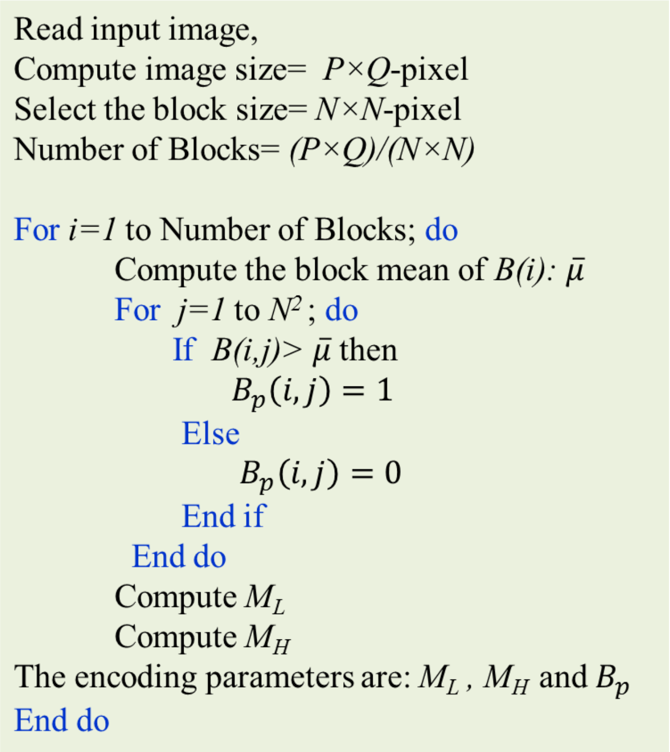
where M_L and M_H represent the low and high means respectively, and N_L and N_H are the number of pixels that are less and greater than the threshold, respectively.

Further, the decoding process is carried out by replacing each element in the bit plane by its corresponding low or high mean.

$$I_{decoded} = \begin{cases} M_L \ if\ B_p(i,j) = 0 \\ M_H \qquad otherwise \end{cases} \tag{3.7}$$

where $I_{decoded}$ is the reconstructed image.

Further, the AMBTC algorithm computation procedure is shown in Figure 3.1.



```
Read input image,
Compute image size= $P\times Q$-pixel
Select the block size= $N\times N$-pixel
Number of Blocks= $(P\times Q)/(N\times N)$

For $i=1$ to Number of Blocks; do
        Compute the block mean of $B(i)$: $\bar{\mu}$
        For $j=1$ to $N^2$; do
            If $B(i,j)>\bar{\mu}$ then
                $B_p(i,j) = 1$
            Else
                $B_p(i,j) = 0$
            End if
          End do
        Compute $M_L$
        Compute $M_H$
The encoding parameters are: $M_L$ , $M_H$ and $B_p$
End do
```

Figure 3.1: The AMBTC algorithm computation procedure.

3.2.2 Adaptive Quantisation Coding (AQC)

AQC is used in different applications as a low complexity and high speed image compression system, such as an image compression system to reduce the motion blur on liquid crystal displays (LCDs) [4, 5, 43-46]. The AQC can be described as a multilevel BTC encoding system. The computation of the AQC compression algorithm includes the following three steps:

1. Partitioning the input image into non-overlapped blocks of $N \times N$-pixel.
2. The computation of the quantisation step:

$$Q_s = \lfloor (B_m - B_n)/Q_L \rfloor \tag{3.8}$$

where Q_s represents the quantisation step , Q_L is the number of quantisation levels, $B_m = Max(B(i,j))$ and $B_n = Min(B(i,j))$.

3. The bit plane (B_p) of each block using a multilevel quantiser can be computed as follows:

$$B_p(i,j) = \lfloor (B(i,j) - B_n)/Q_s \rfloor \tag{3.9}$$

Hence, the encoded bit stream includes bit plane, the quantiser step and the minimum of each block. These values are termed the AQC parameters. The computation procedure of the AQC algorithms is shown in Figure 3.2.

```
Read input image,
Compute image size= $P \times Q$-pixel
Select the block size= $N \times N$-pixel
Number of Blocks= $(P \times Q)/(N \times N)$
Specify the number of quantisation levels; $Q_L$

For $i=1$ to Number of Blocks; do
        Compute the Maximum intensity value of the block B(i); $B_m$
        Compute the Minimum intensity value of the block B(i); $B_n$
        Compute the quantisation step; $Q_s = \lfloor (B_m - B_n)/Q_L \rfloor$

## Bit plane $B_p$ computation
        For $j=1$ to $N^2$; do
            $B_p(i,j) = \lfloor (B(i,j) - B_n)/Q_s \rfloor$
        End do
The encoding parameters are: $B_n$, $B_p$ and $Q_s$
End do
```

Figure 3.2: The AQC algorithm computation procedure.

The decoding phase is carried out by restoring the original image from the computed AQC parameters, as follows:

$$\tilde{B}(i,j) = B_n + Q_s \times B_p(i,j) \tag{3.10}$$

where, $\tilde{B}(i,j)$is the reconstructed image block.

As such, the resultant bit rate of a grey scale image is expressed as follows:

$$BitRate = \frac{(Bit_{Qs} + Bit_n)}{N \times N} + log_2 Q_L \tag{3.11}$$

where Bit_{Qs} is the number of bits required for the quantiser step, which is equal to $8 - log_2 Q_L\ bits$. $log_2 Q_L$ is the quantisation level bits and Bit_n is the number of bits used for encoding the minimum of each block (typically 8 bits). As an example, if a 512×512 grey scale image is partitioned into 4×4-pixel blocks, the resultant bit rates are equal to 1.8125 and 3.8125 bpp using two and eight levels quantisers, respectively.

The AQC algorithm outperforms the BTC algorithm in terms of the quality of the output images, but with a higher bit rate than the BTC algorithm [4]. However, the AQC algorithm produces a fixed bit rate throughout the whole image without considering the difference in intensity levels between image regions. Some image regions contain a single intensity level, whereas the others contain different intensity levels. Hence, improved encoders are proposed by considering the intensity variation of the image blocks, namely: the IBAQC.

3.3 Intensity Based Adaptive Quantisation Coding (IBAQC)

In this section, two IBAQC image compression algorithms are presented. The proposed algorithms consider the intensity variation of image regions to reduce the bit rates whilst keeping the quality of the compressed image at an acceptable level. Both algorithms comprise two steps: intensity check and AQC computation. The intensity check step is carried out locally within each data block using a certain intensity check unit. The two algorithms are termed Algorithm1 and Algorithm2. Algorithm1 performs the intensity check using the block variance prior to the AQC computation process. While, Algorithm2 performs the intensity check on the quantisation step of the AQC algorithm. The input parameters of both algorithms are the number of quantisation levels, block size and threshold.

3.3.1 Algorithm1

The first algorithm (Algorithm1) is a composition of intensity variation check and AQC units, as shown in Figure 3.3 and Figure 3.4 . The input image is partitioned into non-overlapping $N \times N$-pixel blocks to check the local variation of each image block using the sample variance ($\overline{\sigma^2}$). The variance can be computed using (3.4). A block variance is compared against a predefined threshold to classify the image blocks into low or high intensity variation. The variance of a small image block is small enough to ignore the bit plane of the whole block. Thus, an appropriate encoding method and the parameters for each image block are selected. The encoding parameters of the two classes of image blocks can be described as follows:

$$I_{out} = \begin{cases} \{\rho, B_n, B_p, Q_s\} & if\ \overline{\sigma^2} > \delta \\ \{\rho, \bar{\mu}\} & otherwise \end{cases} \tag{3.12}$$

where I_{out} is the encoded bit stream, $\rho \in \{0,1\}$ is the classification bit and δ is an adjustable parameter which represents a predefined threshold. Further, the number of bits required for the quantisation step is given as follows:

$$Bit_{Qs} = \left\lceil log_2\Big(Max\big(B(i,j)\big)\Big)\right\rceil - log_2 Q_L \tag{3.13}$$

where $\lceil : \rceil$ represents ceil function.

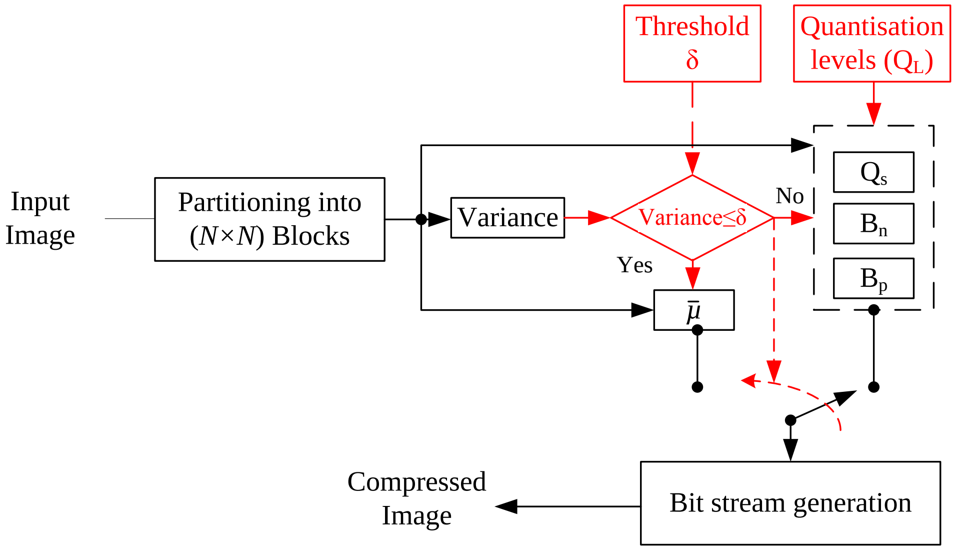


Figure 3.3: Block diagram of Algorithm1.

```
Read input image,
Compute image size  P×Q-pixel
Select the block size N×N-pixel
Number of Blocks= P×Q/( N×N)
Specify the number of quantisation levels; $Q_L$
Specify the classification threshold; $\delta$

For $i=1$ to Number of Blocks; do
        Compute the Mean value of the block B(i); $\bar{\mu}$
        Compute the variance of the block $B(i)$; $\overline{\sigma^2} = \overline{\mu^2} - \bar{\mu}^2$
    If  $\overline{\sigma^2} > \delta$ then
        Compute the Maximum intensity value of the block B(i); $B_m$
        Compute the Minimum intensity value of the block B(i); $B_n$
        Compute the quantisation step; $Q_s = \lfloor (B_m - B_n)/Q_L \rfloor$

    ## Bit plane $B_p$ computation
         For  $j=1$ to $N^2$ ; do
             $B_p(i,j) = \lfloor (B(i,j) - B_n)/Q_s \rfloor$
         End do

         Set $\rho=1$
         The encoding parameters are:  $\rho$, $B_n$, $B_p$ and $Q_s$
     Else
        Set $\rho=0$
        The encoding parameters are: $\rho$ and $\bar{\mu}$
    End if

End do
```

Figure 3.4: The Computation procedure of Algorithm1.

Typically, the number of quantisation step bits is equal to $(8 - \log_2 Q_L)$ in typical 8 bit/pixel grey images. The number of bits required for the bit plane of high variation blocks is:

$$Bit_{Bp} = (N \times N) \log_2 Q_L \tag{3.14}$$

Thus, the bit rate of the proposed algorithm for each image block is:

$$BitRate = \begin{cases} \dfrac{(1 + Bit_{\mu})}{N \times N} & if\, \overline{\sigma^2} < \delta \\ \dfrac{(1 + Bit_{Qs} + Bit_n + Bit_{Bp})}{N \times N} & otherwise \end{cases} \tag{3.15}$$

where Bit_{μ}, Bit_{Qs}, Bit_n and Bit_{Bp} is the number of bits required to represent; the mean, quantisation step, minimum and the bit plane of each block, respectively. Overall, the number of bits that are required to encode the low and high intensity variation blocks for 4×4-pixel blocks are shown in Table 3.1. From Table 3.1, a reduction of 72-88 % in bit rate can be achieved for the low intensity variation blocks. Thus, the low intensity variation blocks are encoded using its mean and a 1 bit classifier, whereas the edge blocks are encoded using its minimum, quantisation step and bit plane. The number of quantisation levels can be set to 3, 4 or 5 depending on the required compression ratio and the output performance.

The output of the former computation procedure represents the encoded or compressed image. On the decoding side, the decoder takes account of the classifier bit (ρ). If it is in low state, the block is classified and decoded as a low variation block; otherwise, it is decoded as a high variation block. Thus, the decoding phase can be represented by the following equation:

$$\tilde{B}(i,j)=\begin{cases} B_n+B_p(i,j)\times Q_s & if\ \rho=1 \\ \bar{\mu} & otherwise \end{cases} \tag{3.16}$$

where $\tilde{B}(i,j)$is the decompressed image block.

Table 3.1: Number of bits required for each block type for 4×4-pixel blocks using different number of quantisation levels.

Quantisation levels (Q_L)	Classifier bit	Bit_n or Bit_μ	Bit_{Qs}	Bit_{Bp}	Number of bit/Block for each intensity variation type:		Bit rate Saving/Block %
					High	Low	
2	1	8	7	16	32	9	72
4	1	8	6	32	47	9	80
8	1	8	5	48	62	9	85
16	1	8	4	64	77	9	88

3.3.2 Algorithm2

This algorithm is a composition of AQC and an intensity variation check unit, as shown in Figure 3.5 and Figure 3.6. The input image is partitioned into $N\times N$-pixel blocks in which the AQC parameters (the minimum of each block; B_n, quantisation step; Q_s and the bit plane; B_p) are computed. The AQC parameters and the output bit rate depend on the predefined quantiser, as stated in (3.9) and (3.11). The quantiser is selected according to the target bit rate and the required PSNR.

The proposed algorithm classifies each image block into one of two classes, low or high intensity variation block. High intensity variation blocks are encoded using the AQC coding system while low ones are represented by its mean, similar to Algorithm1. However, in Algorithm1 the intensity check unit utilises high computation complexity, which involves $N^2 + 1$ multiplication and $2(N^2 - 1)$ additions to perform the block variance computation. Thus, to eliminate this number of multiplications a new intensity check unit is proposed. The new intensity check unit is based on the quantisation step of the quantiser, as the low variation blocks require lower quantisation steps than the high variation or edge blocks. Thus, to exploit this attribute for bit rate reduction, the quantisation step is compared with a predefined threshold. The bit plane and quantisation steps are discarded for any block for which the quantisation step is less than the predefined threshold. Thus, each low intensity variation block is represented by its mean and one classifier bit to distinguish between the two types of blocks. Moreover, the encoding bit stream of the high intensity variation blocks consists of a bit plane, a quantiser step size, the minimum intensity value of such a block and a single bit classifier. The encoded parameters of both block classes can be expressed as in (3.12) and the output bit rate is computed using (3.15), while the decoder described in (3.16) can be used in the decoding phase.

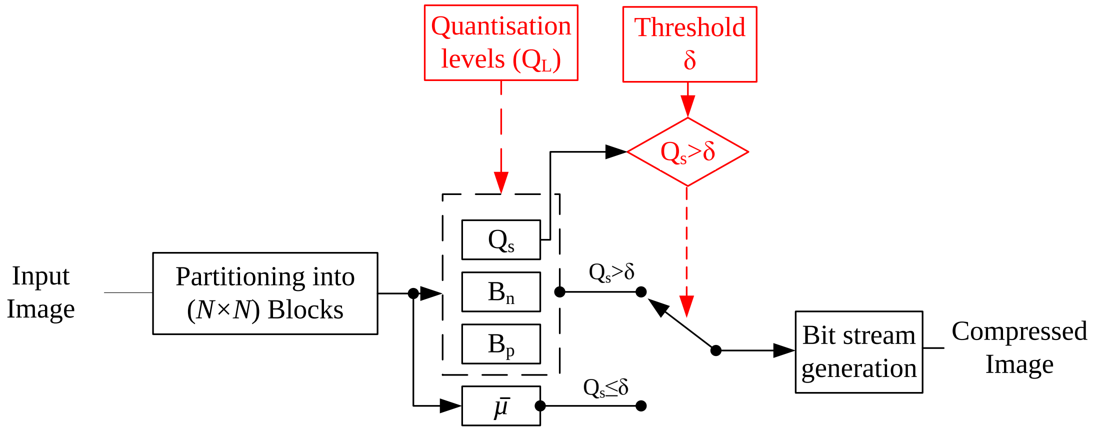


Figure 3.5: Block diagram of Algorithm2.

```
Read input image,
Compute image size  P×Q-pixel
Select the block size N×N-pixel
Number of Blocks= P×Q/( N×N)
Specify the number of quantisation levels; $Q_L$
Specify the classification threshold; δ

For $i=1$ to Number of Blocks; do
       Compute the Mean value of the block B(i); $\bar{\mu}$
       Compute the Maximum intensity value of the block B(i); $B_m$
       Compute the Minimum intensity value of the block B(i); $B_n$
        Compute the quantisation step; $Q_s=\lfloor (B_m - B_n)/Q_L \rfloor$

        If  $Q_s > \delta$ then

        ## Bit plane $B_p$  computation
               For  $j=1$ to $N^2$ ; do
                    $B_p(i,j)=\lfloor (B(i,j) - B_n)/Q_s \rfloor$
              End do

              Set $\rho=1$
              The encoding parameters are:  $\rho, B_n, B_p$ and $Q_s$
         Else
             Set $\rho=0$
             The encoding parameters are: $\rho$ and $\bar{\mu}$
        End if

End do
```

Figure 3.6: The Computation procedure of Algorithm2.

3.4 The Proposed IBAQC Architectures

In this section, two architectures that implement the proposed IBAQC algorithms are presented. The two architectures are named Model1 and Model2, which represent the architectures that corresponding to Algorithm1 and Algorithm2, respectively. Further, an architecture for the AQC algorithm is also designed and verified. The red and blue lines represent the controller signals while the black lines represent the actual data flow in the proposed architectures. The proposed architectures have been designed and implemented using Xilinx System generator tool, as illustrated in section 2.8. The initialisation Matlab codes for the proposed architectures are listed in the appendix B.

3.4.1 Model1

The proposed Model1 architecture consists of five separate units: the main controller, the intensity check, the minimum, maximum and AQC computation units, as shown in Figure 3.7. Model1 architecture represents the hardware architecture of Algorithm1.

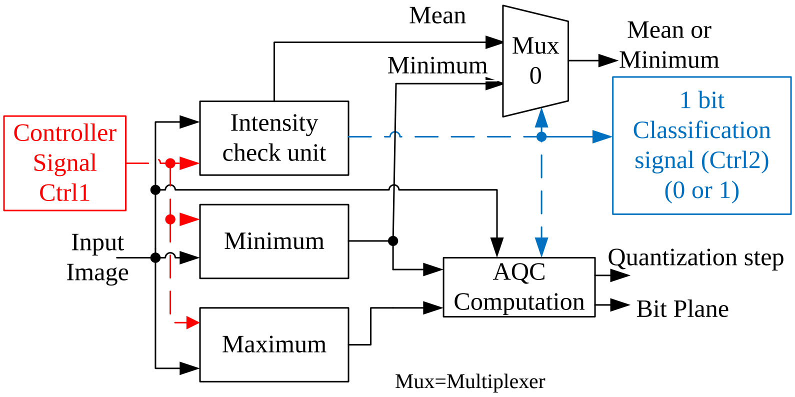


Figure 3.7: IBAQC Model1 architecture.

The proposed architecture produces four outputs: the intensity check signal value Ctrl2, the minimum or mean of each block, the bit plane and quantisation step. Ctrl2 is a binary signal which distinguishes between the high and low intensity blocks (the classifier bit).

3.4.1.1 Minimum, Maximum and Mean Computation Units

The computation of the minimum and maximum of each input block is carried out using specific circuits composed of comparators, multiplexers and registers, as shown in Figure 3.8. The input parameters to the proposed architectures, which are the block size, number of quantisation levels, threshold and wordlengths are specified prior to the computation process of the compressed image.

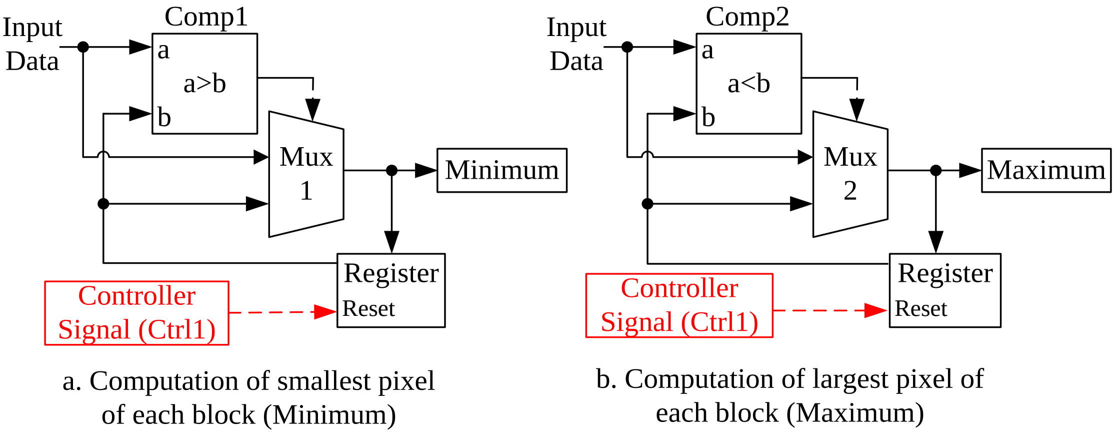


Figure 3.8: Minimum and Maximum computation units.

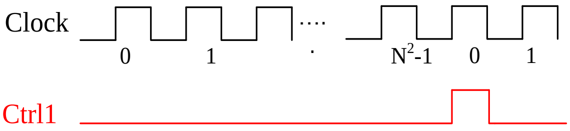


Figure 3.9: Controller signal (Ctrl1).

The minimum value of each block is computed as follows: each input pixel is compared with the output of the multiplexer Mux1, as shown in Figure 3.8-a. The output of the comparator Comp1 is high if the input pixel is greater than the Mux1 output; otherwise, it is low. The Comp1 output acts as a selector signal for the Mux1; if it is 1, the output of Mux1 is unchanged; otherwise, Mux1 selects the current input pixel. After N^2 clock cycles, the output of Mux1 represents the minimum value of the data block and the register is set to its initial value (255) by the controlling signal Ctrl1. The maximum value of each input block is computed in the same way except for the comparison operation and the initial value of the register is zero. The output of Comp2 is high when the input data is less than Mux2 output. The block diagram of the maximum computation circuit is shown in Figure 3.8-b.

Further, the mean of each input block is computed through the computation process of the intensity check unit. The controller signal Ctrl1 is used to reset the accumulators, enable/disable multipliers and set/reset registers. This signal is generated according to the size of input block and its waveform is shown in Figure 3.9.

3.4.1.2 Intensity Check Unit

The intensity check unit is a combination of multipliers, accumulators (Accum), bit shifting and a comparator (Comp), as shown in Figure 3.10.

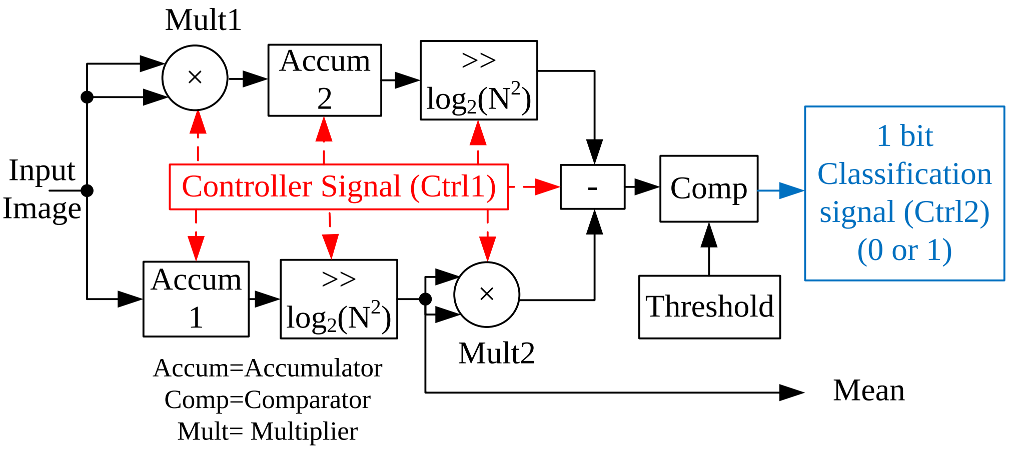


Figure 3.10: Intensity check unit of Model1.

During each N^2 clock cycles, the following sequence of operations is carried out in the intensity check unit:

- The sum of all pixels in the $N \times N$-pixel block using Accum1, as shown in Figure 3.10
- The computation of the mean of each block ($\bar{\mu}$) using a $log_2(N^2)$ right bit shifter
- The computation of the square value of the mean ($\bar{\mu}^2$) using Mult1
- The computation of the squared value of the mean of each input block using Mult2
- The sum of the squared values of all pixels in the $N \times N$-pixel block using Accum2
- The computation of the mean of squared values $\overline{\mu^2}$ of each block using a $log_2(N^2)$ right bit shifter
- The computation of the variance, is given by:

$$\overline{\sigma^2}=\overline{\mu^2} - \bar{\mu}^2 \tag{3.17}$$

The last computation step of the intensity check unit is the comparison between the variance and a predefined threshold to generate the bi-level controlling signal Ctrl2.

Such a control signal is low for low intensity variation blocks Ctrl2=0. Moreover, the AQC computation is accomplished using the circuit of Figure 3.11. The AQC computation unit is turned on/off according to the state of Ctrl2 signal; if Ctrl2=0, the quantisation step and the bit plane are set to 0. The controller signal Ctrl2 is used to select the minimum or the mean of each block by driving Mux0 in Figure 3.7.

3.4.1.3 AQC Computation Unit

The next steps are the computation of the bit plane and the quantisation step of the image block. The bit plane is computed according to (3.8). However, division operations are avoided using two 256-Word ROMs and an addressing unit, as shown in Figure 3.11. The difference between the maximum and minimum ranges is from 0 to 255, which can be used as the addresses of the two ROMs (ROM1 and ROM2). Thus, the content of ROM1 is:

$$Q_s = \left\lceil \frac{0:255}{Q_L} \right\rceil \tag{3.18}$$

and the content of ROM2 is given as:

$$Q_s^{-1} = \frac{1}{Q_s}, Q_s^{-1}(1) = 0 \tag{3.19}$$

where Q_s is the quantisation steps and $\lceil \quad \rceil$ represents the ceil function and Q_s^{-1} is an array of 256 words used for the bit plane computation in (3.9). The output from ROM2 is multiplied by the difference between the actual pixel value and the minimum of each block to produce the bit plane of the input data block. Such a bit plane is coded as unsigned integer on $log_2 Q_L$ bit.

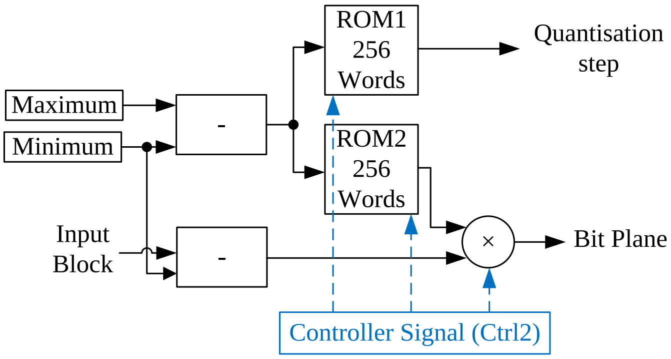


Figure 3.11: AQC computation unit of Model1.

3.4.2 Model2

Model2 architecture consists of six separate units: the main controller, intensity check, minimum, maximum, mean and AQC computation units, as shown in Figure 3.12. The input parameters of the proposed architecture are the quantisation levels, block size and the threshold value.

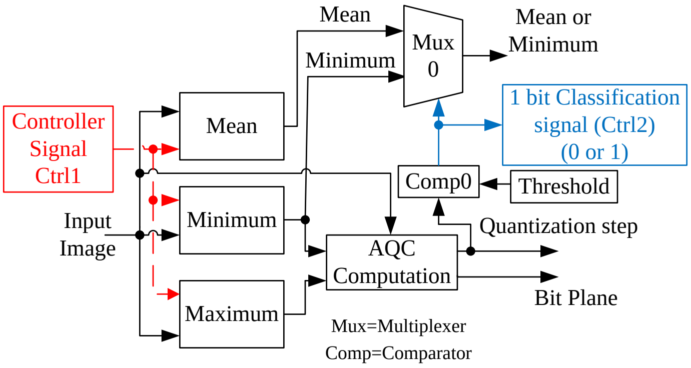


Figure 3.12: IBAQC Model2 architecture.

The computation of the minimum and maximum of each input block is carried out using the same procedure of Model1, while the mean of each input block is computed using an accumulator and $log_2 N^2$ right bit shifter. The AQC parameters are computed using the circuit that shown in Figure 3.13. The computation process is carried out similar to that of Model1 architecture. However, the ROM1 output is compared against the predefined threshold to generate the second controlling signal (Ctrl2), as shown in Figure 3.13. When Ctrl2 is 0, the multiplier is turned off, the output of Mux3 is set to 0 and the mean of each block is selected, otherwise the multiplier is on, the Mux3 output is set to Q_s and the minimum of each input block is selected. The Q_s, Mult1 output (bit plane), minimum and the Ctrl2 binary signal represent the encoded bit stream of the high intensity variation input block.

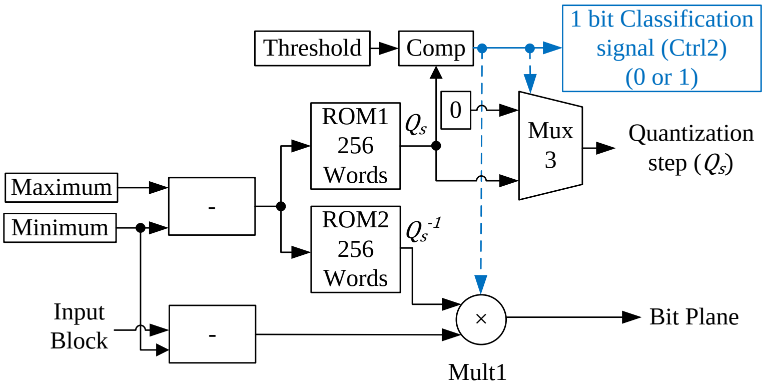


Figure 3.13: AQC computation unit of Model2.

3.4.3 AQC Architecture

The AQC architecture is designed similar to Model2 architecture, with only difference being the intensity check unit which has been removed. The block diagram of the AQC architecture is shown in Figure 3.14 and its operation sequence is similar to Model2 architecture. However, all image blocks are encoded using the minimum, bit plane and the quantisation step.

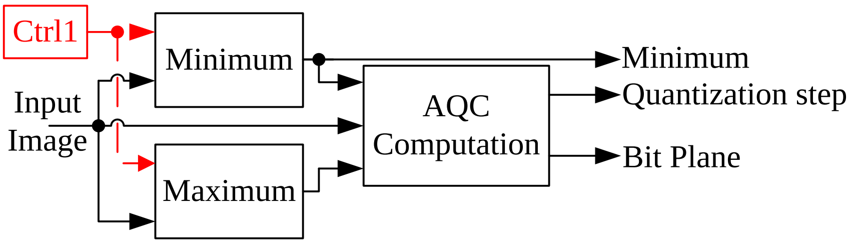


Figure 3.14: The AQC architecture.

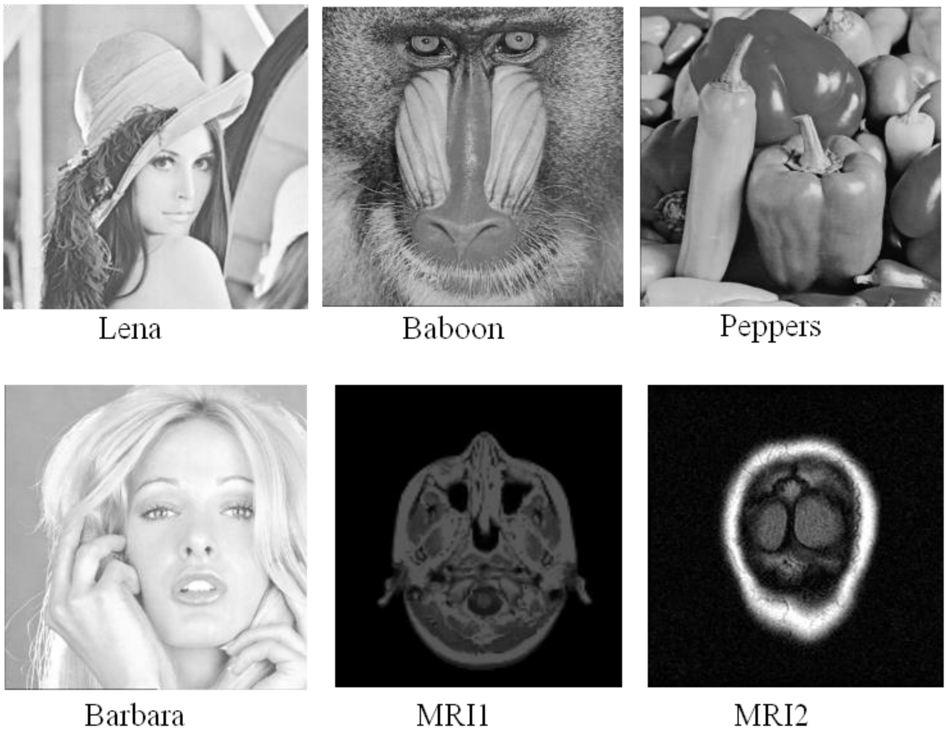
Further, the computational complexities and the initial latencies of the AQC, Model1 and Model2 architectures are shown in Table 3.2. It can be seen from Table 3.2, the AQC and Model2 architectures have the same number of multiplication operations. While additional $N^2 - 1$-addition operations are required to perform the block mean computation in Model2. Further, Model1 shows an additional number of multiplication and addition operations over AQC and Model2 which are required for variance computation. However, the number of addition and multiplication operations depends on the type of each image block. For high intensity variation blocks; $3N^2 - 1$ and $2N^2 + 1$ addition and multiplication operations are required, respectively. Whereas only $2(N^2 - 1)$and $N^2 + 1$ addition and multiplication operations are required for low intensity variations blocks.

Table 3.2: Computational complexities of the AQC, Model1 and Model2 architectures.

Architecture	Latency	Number of	
		Additions	Multiplications
AQC	$2N^2 + 6$	$N^2 + 1$	N^2
Model1	$2N^2 + 10$	$3N^2 - 1$: (for high intensity variation blocks) or $2(N^2 - 1)$:(for low intensity variation blocks)	$2N^2 + 1$:(for high intensity variation blocks) or $N^2 + 1$:(for low intensity variation blocks)
Model2	$2N^2 + 6$	$2N^2$	N^2

3.5 Performance Evaluation

In this section, the proposed algorithms and their corresponding architectures are tested on natural and commonly used images, as shown in Figure 3.15. Tests have been carried out using different block sizes, quantisation levels and thresholds. The block sizes used are: 4×4, 8×8 and 16×16-pixel while the quantisation levels are 8, 16 and 32 levels.



Lena Baboon Peppers

Barbara MRI1 MRI2

Figure 3.15: Test images.

3.5.1 Performance of the Proposed Algorithms

The obtained results are evaluated using the bit rate and PSNR of the reconstructed images. A Matlab code is written to compute the PSNR using (2.13).

3.5.1.1 Rate Distortion Performance of Algorithm1

The PSNR and bit rates of Algorithm1 using low threshold value of 12 and high threshold value of 400 are shown in Table 3.3 and Table 3.4, respectively. The low threshold (Table 3.3) produces very good image quality and high average PSNRs ranging from 41 to 46 dB with a bit rate of 3 to 4 bpp. As can be seen from Table 3.3, the best choice of the block size is 4×4-pixel with an 8-level quantiser. Such a block size, quantiser and threshold parameters have resulted in a good compromise between image quality and bit rate.

Table 3.3: PSNR and bit rate computed using Algorithm1 for different block sizes, quantisation levels and a low threshold (12).

		Lena (512×512)		Baboon (512×512)		Pepper (512×512)		Barbara (512×512)		MRI1 (128×128)		MRI2 (256×256)	
N	Q_L	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)
4	32	43.9	3.3	48.9	5.6	44.3	4.2	43.8	3.3	53.7	2.5	48.4	5.3
4	16	43.4	2.8	44.0	4.7	43.5	3.5	43.2	2.8	49.7	2.2	46.0	4.4
4	8	41.4	2.3	37.4	3.8	40.5	2.9	40.9	2.3	41.3	1.8	41.6	3.6
8	32	44.6	3.6	47.4	5.2	45.6	4.4	44.7	3.6	54.1	2.3	48.7	5.0
8	16	43.2	2.9	42.0	4.2	43.3	3.6	43.3	2.9	48.0	1.9	43.7	4.1
8	8	39.3	2.2	35.7	3.2	38.3	2.7	39.4	2.3	38.1	1.5	37.5	3.1
16	32	45.0	4.1	45.8	5.0	46.0	4.7	45.7	4.2	52.9	2.8	47.2	5.0
16	16	42.2	3.3	40.3	4.1	41.7	3.8	42.8	3.4	47.3	2.2	41.0	4.1
16	8	36.9	2.5	34.2	3.1	35.6	2.9	37.4	2.5	36.7	1.7	34.1	3.1
Average		**42.2**	**3.0**	**41.7**	**4.3**	**42.1**	**3.6**	**42.4**	**3.0**	**46.8**	**2.1**	**43.1**	**4.2**

Table 3.4: PSNR and bit rate computed using Algorithm1 for different block sizes, quantisation levels and a high threshold (400).

		Lena (512×512)		Baboon (512×512)		Pepper (512×512)		Barbara (512×512)		MRI1 (128×128)		MRI2 (256×256)	
N	Q_L	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)
4	32	31.8	0.9	29.5	2.8	32.1	1.0	32.3	0.9	31.4	1.3	30.9	1.2
4	16	31.8	0.8	29.5	2.4	32.1	0.9	32.3	0.8	31.3	1.1	30.8	1.1
4	8	31.7	0.8	29.0	2.0	31.9	0.9	32.1	0.7	31.1	1.0	30.7	1.0
8	32	30.9	1.0	29.9	3.0	30.9	1.1	30.7	0.7	31.7	1.5	31.4	1.2
8	16	30.8	0.8	29.7	2.4	30.9	0.9	30.6	0.6	31.6	1.3	31.3	1.0
8	8	30.5	0.6	29.1	1.9	30.4	0.7	30.4	0.5	30.9	1.0	30.6	0.8
16	32	30.2	1.6	30.6	3.4	30.4	1.9	28.9	1.0	32.0	2.0	32.5	1.5
16	16	30.1	1.3	30.3	2.7	30.3	1.5	28.9	0.8	31.9	1.6	32.1	1.2
16	8	29.6	1.0	29.2	2.1	29.5	1.2	28.5	0.6	30.9	1.2	30.4	0.9
Average		**30.8**	**1.0**	**29.6**	**2.5**	**30.9**	**1.1**	**30.5**	**0.7**	**31.4**	**1.3**	**31.2**	**1.1**

Furthermore, Table 3.4 shows the performance of the encoded images using a high classification threshold. The average bit rates, in this case, are 1 to 2.5 bpp with average PSNRs of 30 to 31 dB. All the images behaved equally except the baboon image, which is an example of a high intensity variation image. Further, the 4×4-pixel block size and 16-level quantiser parameters provide the best compromises between image quality and bit rate in the case of a high classification threshold.

3.5.1.2 Rate Distortion Performance of Algorithm2

Samples of the obtained results using Algorithm2 are shown in Table 3.5 and Table 3.6 as an illustration of low and high thresholds cases, respectively. Table 3.5 shows the PSNRs and bit rates obtained using a low classification threshold for very good output image quality. The blocks which have a quantisation step of less than 0.4 are dropped, and classified as low intensity variation blocks and encoded by their mean and a single bit classifier. The remaining high intensity variation blocks are encoded using their minimum, quantisation step, bit plane and a single bit classifier. From Table 3.5, the average PSNRs of the test images range from 41 to 47 dB and the average corresponding bit rates are 2.2 to 4.4 bpp. Further, for such high output accuracy, it can be concluded that the block size of 4×4-pixel with an 8-level quantiser (shaded cells in Table 3.5) produces the best compromises between the output PSNRs and bit rates.

Furthermore, Table 3.6 presents the PSNRs and bit rates for a classification threshold of 2. Such a high threshold has led to more blocks being classified as low intensity variation and represented by their means and a single bit classifier, which consequently has resulted in low bit rate encoded image. From this table, the block size of 4×4-pixel and 32 quantisation levels have resulted in a best PSNR for low bit rate purposes. The average PSNRs in this case are 30 to 32 dB and the bit rate is around 1 bpp. However, in the case of the baboon image, such PSNR can only be attained on a higher bit rate. This is due to the fact that this particular image has many regions with high intensity variations.

Table 3.5: PSNR and bit rate computed using Algorithm2 for different block sizes, quantisation levels and a low threshold (0.4).

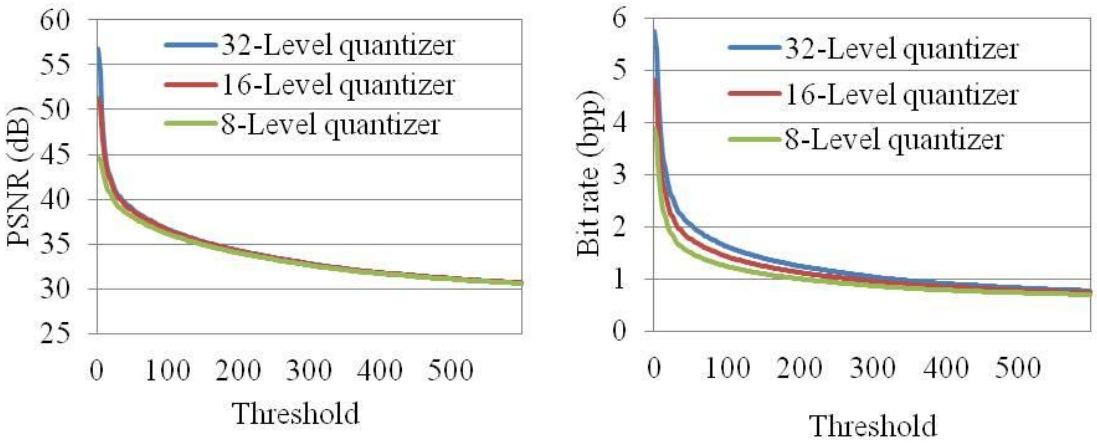
N	Q_L	Lena (512×512)		Baboon (512×512)		Pepper (512×512)		Barbara (512×512)		MRI1 (128×128)		MRI2 (256×256)	
		PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)
4	32	43.8	3.31	48.9	5.7	44.4	4.2	43.7	3.3	53.9	2.5	47	5.2
4	16	48.9	4.12	44.4	4.8	49.4	4.7	49.1	4.3	50.4	2.2	48.5	4.8
4	8	44.6	3.82	37.5	3.9	42.5	3.9	43.7	3.8	41.4	1.9	42.3	3.9
8	32	47.1	4.09	47.5	5.2	49.3	4.9	46.7	4.1	54.3	2.4	50.1	5.2
8	16	47	4.11	42	4.2	45.7	4.2	47.5	4.1	48.1	1.9	44.2	4.2
8	8	40.5	3.22	35.7	3.2	38.9	3.2	40.6	3.2	38.1	1.5	37.6	3.2
16	32	48.3	4.74	45.8	5	47.8	5	49.2	4.8	53.8	2.9	47.2	5
16	16	44	4.04	40.3	4.1	42.3	4.1	44.6	4	47.5	2.4	41	4.1
16	8	37.4	3.05	34.2	3.1	35.8	3.1	37.8	3.1	36.7	1.8	34.1	3.1
Average		**44.6**	**3.8**	**41.8**	**4.4**	**44.0**	**4.1**	**44.8**	**3.9**	**47.1**	**2.2**	**43.6**	**4.3**

Table 3.6: PSNR and bit rate computed using Algorithm2 for different block sizes, quantisation levels and a high threshold (2).

N	Q_L	Lena (512×512)		Baboon (512×512)		Pepper (512×512)		Barbara (512×512)		MRI1 (128×128)		MRI2 (256×256)	
		PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)
4	32	31.9	0.9	30.4	3	32.3	1	32.2	0.9	31.3	1.3	31.2	1.2
4	16	36.8	1.5	37	3.7	35.7	1.5	36.3	1.3	41.5	1.9	34.6	1.7
4	8	39.9	1.9	37.2	3.7	38.7	2.3	39.6	1.9	41.2	1.8	39.2	2.9
8	32	32.4	1.2	33.5	3.6	32.5	1.4	31.9	0.9	35.3	1.9	33.2	1.4
8	16	38.0	1.8	39.3	3.8	37	2.1	37.5	1.8	47.8	1.9	34.7	1.6
8	8	39.2	2.2	35.7	3.2	38.2	2.7	39.2	2.2	38.1	1.5	37.4	3.1
16	32	33.8	2.2	36.4	4.3	33.8	2.6	32.6	1.9	48.9	2.7	34.2	1.7
16	16	39.2	2.6	40.1	4	39.2	3.2	39.2	2.7	47.3	2.2	34.8	1.9
16	8	37.1	2.6	34.2	3.1	35.7	3	37.5	2.6	36.7	1.7	34.1	3.1
Average		**36.5**	**1.9**	**36.0**	**3.6**	**35.9**	**2.2**	**36.2**	**1.8**	**40.9**	**1.9**	**34.8**	**2.1**

3.5.1.3 Rate Distortion Performance Comparison

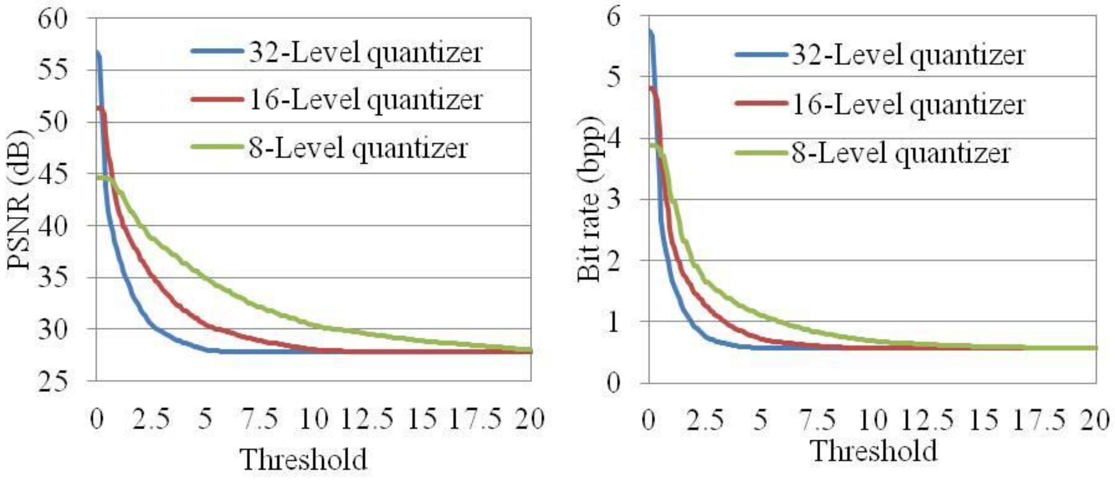
The bit rate and PSNR performance of both algorithms are almost similar. However, the main difference between these two algorithms is the complexity of the intensity check unit. Such a unit requires $N^2 + 1$ multiplication and $2(N^2 - 1)$ addition operations in the first algorithm, whereas the second algorithm involves only $N^2 - 1$ addition operations which are required to compute the mean of the block. Furthermore, by suitably adjusting thresholds, both algorithms can produce the same performance as shown in Figure 3.16 and Figure 3.17. These figures present the PSNRs and bit rates obtained for the Lena image using the proposed algorithms for a 4×4-pixel block size, and 32, 16 and 8 quantisation levels. In Figure 3.16, the sample variance of each image block is compared against a specific set of threshold values, whereas in Figure 3.17 the quantisation step is compared against a wide range of thresholds. From both figures, different PSNRs and bit rates can be obtained by adjusting thresholds and quantisation levels for specific block sizes. The same behaviour is obtained using 8×8 and 16×16-pixel block sizes for the other test images.



a. PSNR (dB) versus Threshold

b. (Bit rate (bpp) versus Threshold

Figure 3.16: Performance of the Lena image obtained using Algorithm1 for a block size of 4×4-pixel, different thresholds and 32, 16 and 8 quantisation levels.



a. PSNR (dB) versus Threshold.

b. Bit rate (bpp) versus Threshold.

Figure 3.17: Performance of the Lena image obtained using Algorithm2 for block size of 4×4-pixel, different thresholds and 32, 16 and 8 quantisation levels.

Further, a comparison between the AQC and the proposed algorithms is shown in Table 3.7. From this table, the output performance of both algorithms outperforms the AQC in terms of bit rate and PSNR. This is due to using a threshold as an additional parameter to control the output bit rate, in contrast with the AQC algorithm which depends on the number of quantisation levels and block sizes only. Thus, the lowest bit rate of the AQC algorithm is 1.1 bpp with a PSNR of 21.6 dB. This result can only be achieved by reducing the number of quantisation levels and increasing the block size. However, bit rates of 0.6 and 0.9 bpp with PSNRs of 30.5 and 31.9 dB are achieved using Algorithm1 and Algorithms2, respectively, using the selected thresholds.

Table 3.7: Performance comparison between the AQC, Algorithm1 and Algorithm2 for Lena image.

AQC algorithm		Algorithm1 Threshold=400		Algorithm2 Threshold=2	
PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)
21.6	1.1	30.5	0.6	31.9	0.9
25.4	1.2	31.8	0.8	32.4	1.2
29.6	1.9	31.8	0.9	36.8	1.5
29.3	2.1	30.9	1	38	1.8
32.9	2.2	30.1	1.3	39.2	2.2
37.1	2.9	30.2	1.6	39.2	2.6

3.5.2 Performance of the Proposed Architectures

The AQC and the proposed architectures are tested on the xc5vlx50t-3ff1136 Virtex 5 FPGA device. The performance of the AQC and the proposed architectures is evaluated using different block sizes, quantisation levels and thresholds. Further, the output accuracy of the proposed architectures are evaluated and compared with the corresponding algorithms' outputs to prove their validity.

3.5.2.1 Hardware Resources

The hardware utilisation rates of the AQC, Model1 and Model2 architectures using various block sizes are shown in Table 3.8. From this table, the average hardware utilisation rates of the proposed architectures are almost equal to 3% of the available resources. However, Model1 architecture consumes slightly more resources due to the requirement of the intensity check unit. The intensity check unit of Model1 requires two additional DSP48Es to compute the variance of each image block. Further, the AQC and Model2 architectures utilised a single multiplier only in their structure.

Table 3.8: Hardware usage of the AQC and the proposed architectures.

Slice Logic Utilisation		4×4-pixel block size			8×8-pixel block size		
	Available	Model1	Model2	AQC	Model1	Model2	AQC
Slice Registers	28,800	280	195	147	341	242	189
Slice LUTs	28,800	225	148	89	286	205	139
Occupied Slices	7,200	97	69	54	126	68	75
Bonded IOBs	480	26	26	25	26	26	25
Block RAM / FIFO (36 Kb)	60	1	1	1	1	1	1
DSP48Es	48	3	1	1	3	1	1
Average (%)		**3**	**2**	**2**	**3**	**2**	**2**

As shown in Table 3.8, additional number of registers, LUTs and occupied slices are required in the case of 8×8-pixel block size. Whereas other hardware resources are remain the same for all block sizes. The extra resources are required due to using extra delay elements in both architectures for synchronisation. Further, the changing in image size and/or the quantisation does not have an impact on hardware utilisation rate.

3.5.2.2 Operating Frequencies and Power Consumption

The operating frequencies, initial latencies and computation times of the AQC and the proposed architectures are shown in Table 3.9. From Table 3.9, a maximum operating frequency of 312, 287 and 292 MHz can be achieved for Model1, Model2 and AQC architectures using 4×4-pixel block size, respectively. Further, the initial latencies of the Model1, Model2 and AQC architectures are: $10 + 2N^2$, $6 + 2N^2$ and $6 + 2N^2$cycles, respectively. Thus, the computation time of a 512×512-pixel image is less than 0.9 ms.

Table 3.9: Maximum operating frequencies, initial latencies and computation time for the AQC and proposed architectures using different block sizes.

	4×4-pixel			8×8-pixel			16×16-pixel		
	Model 1	Model 2	AQC	Model 1	Model 2	AQC	Model 1	Model 2	AQC
Maximum frequency (MHz)	312.0	287.6	292.4	315.9	280.5	285.7	293.8	287.9	283
Initial latency (ns)	134.6	132.1	130	436.8	477.7	469	1776.7	1799.2	1830
Computation time of 512×512 image (ms)	0.84	0.91	0.9	0.83	0.94	0.92	0.89	0.91	0.93

Moreover, the dynamic power consumption of the AQC and the proposed architectures is shown in Figure 3.18. The power consumption is computed using a Xilinx Xpower analyser [125]. The dynamic power consumption of Model1 architecture is higher than that of Model2 architecture by 3 to 5 mW. This power increase with Model1 architecture is due to the complexity of the intensity check unit, which requires two extra multipliers. Further, the power consumption of both the AQC and Model2 architectures is almost the same. For all architectures, an extra 2 mW power is required in the case of 8×8-pixel block size. This extra amount of power is due to using extra hardware resources in this case.

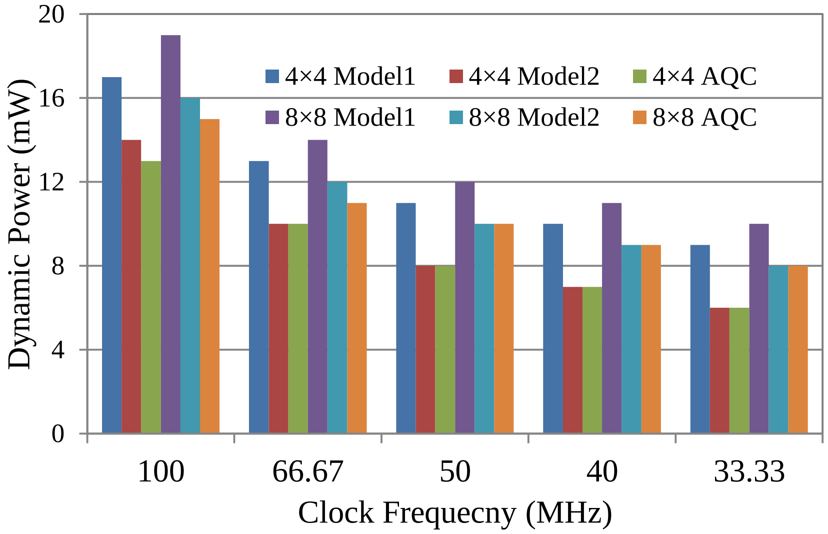


Figure 3.18: Dynamic power consumption of the AQC and the proposed architectures using 4×4 and 8×8-pixel block sizes.

As a result, the performance of Model2 architecture is almost similar to that of the AQC architecture with a few extra hardware resources. The extra hardware resources are used for the mean computation and the intensity check operations. Thus, it can be considered as an appropriate low complexity image compression system with its additional advantage of a good low bit rate-PSNR performance.

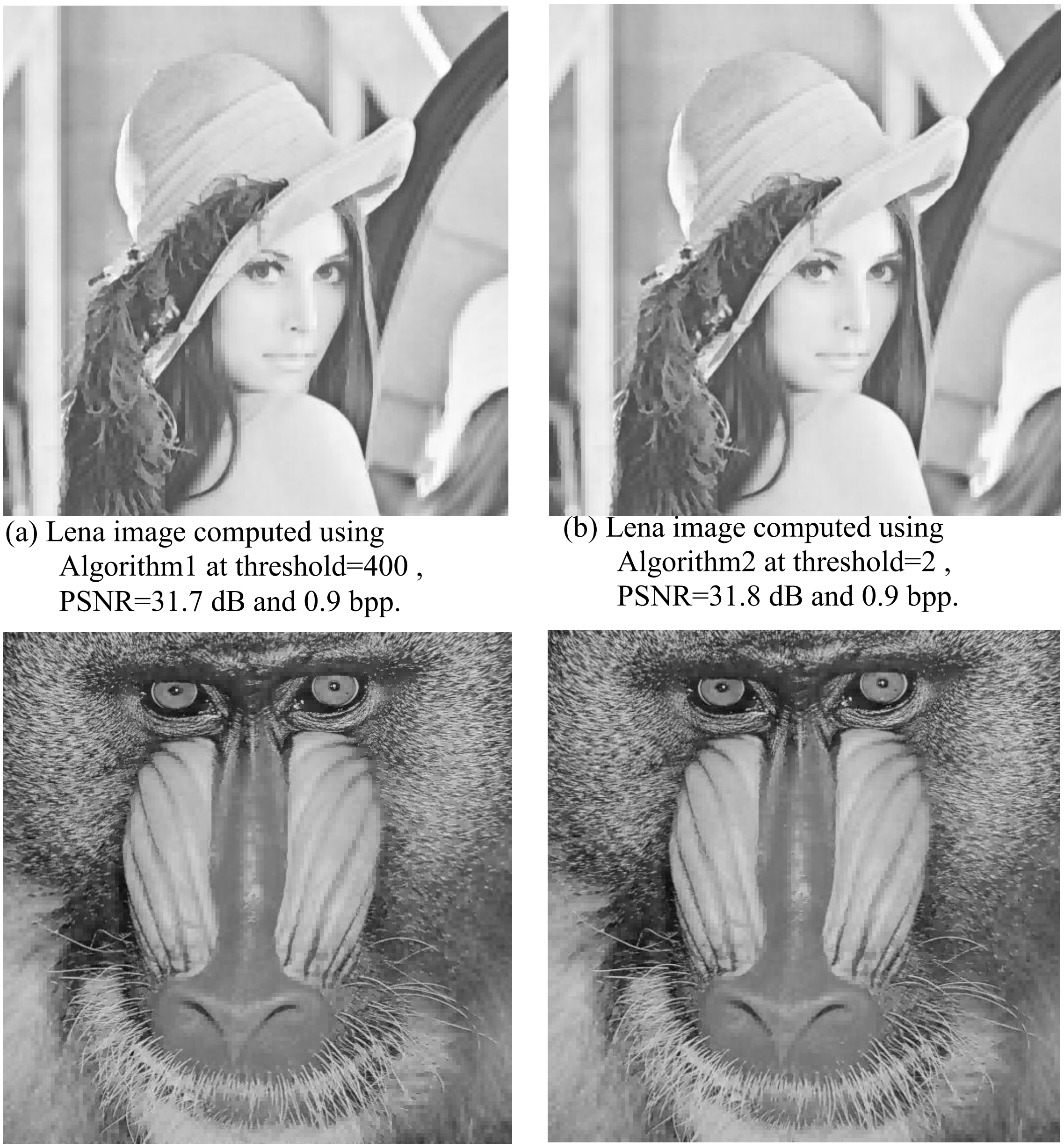
3.5.2.3 The Validation of the Proposed Architectures

The PSNR and bit rate performance of the proposed architectures are evaluated against a Matlab code representation of their respective IBAQC algorithms, as shown in Table 3.10. The drawn figures in this table represent the average PSNR and average bit rate obtained using 8, 16 and 32 quantisation levels, and a 4×4, 8×8 and 16×16-pixel block sizes. Further, from this table the PSNR error between the results obtained by the proposed algorithms and their counterparts' architectures is less than 0.5 dB for some results. This small difference in PSNR is due to the truncation and rounding operations arises from using fixed sizes multipliers in the proposed architectures.

Table 3.10: The average bit rate and PSNR of the proposed algorithms and their corresponding architectures.

Threshold	Lena				Baboon			
	PSNR (dB)	Bit rate (bpp)	PSNR (dB)	Bit rate (bpp)	PSNR (dB)	Bit rate (bpp)	PSNR (dB)	Bit rate (bpp)
	Model1		Algorithm1		Model1		Algorithm1	
12	41.7	3.0	42.2	3.0	41.6	4.3	41.7	4.3
400	30.7	1.0	30.8	1.0	29.4	2.5	29.6	2.5
	Model2		Algorithm2		Model2		Algorithm2	
0.4	44.8	3.9	44.6	3.8	41.8	4.3	41.8	4.4
2	36.5	1.9	36.5	1.9	35.9	3.6	36.0	3.6

Moreover, Lena and the baboon reconstructed images are shown in Figure 3.19 and Figure 3.20. In these two figures, these images are compressed using a 4×4-pixel block size, a 32-level quantiser and a classification threshold of 400 and 2 for Algorithm-Model1 and algorithm2-Model2, respectively. Both algorithms and architectures produced the same image quality regarding to the Lena image (31 dB PSNR with 0.9 bpp bit rate), while there was little difference in the performance of the baboon image. The difference in the performance of the baboon image can be reduced or eliminated by suitably adjusting the classification thresholds in both algorithms and architectures.



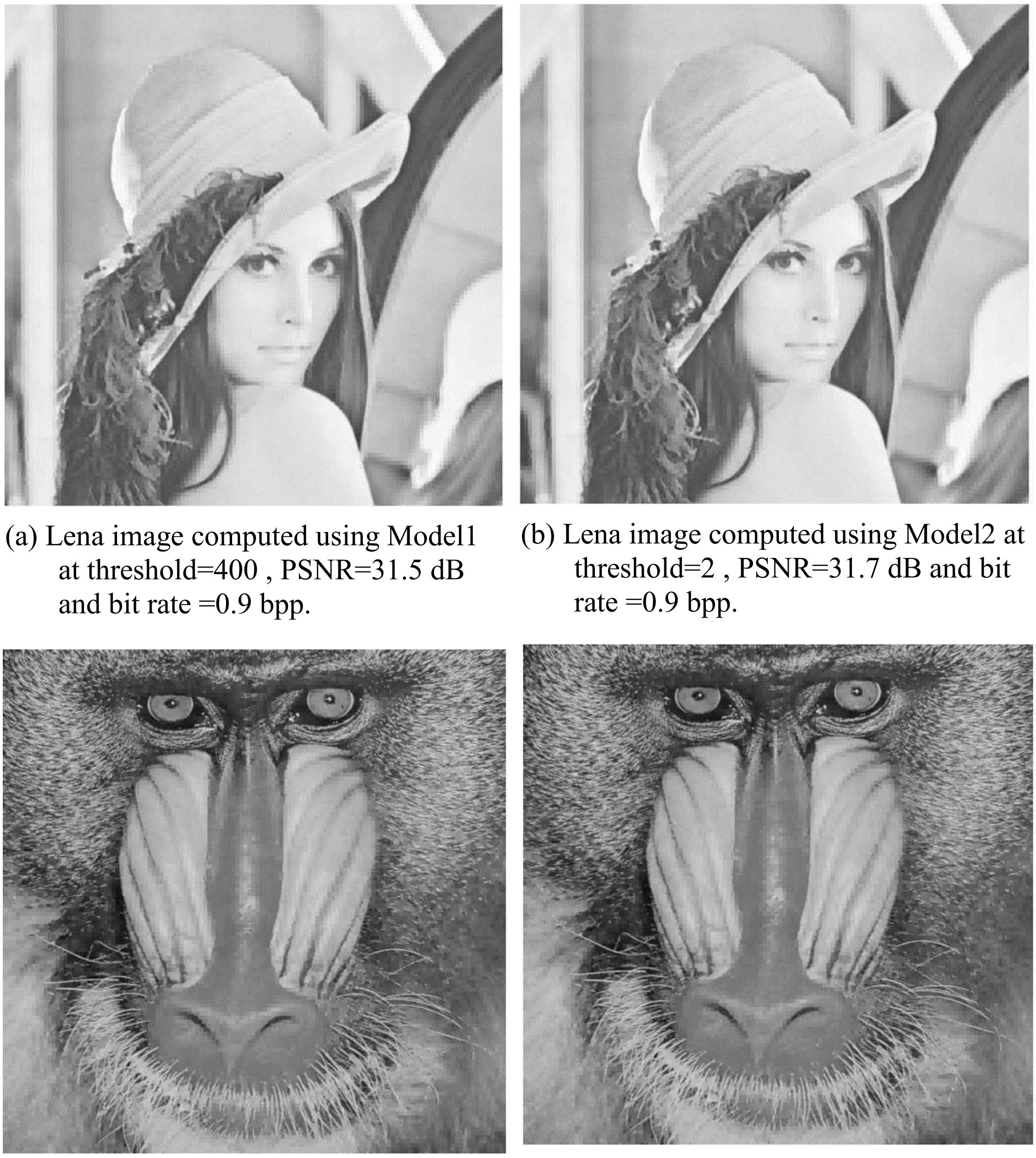
(a) Lena image computed using Algorithm1 at threshold=400 , PSNR=31.7 dB and 0.9 bpp.

(b) Lena image computed using Algorithm2 at threshold=2 , PSNR=31.8 dB and 0.9 bpp.

(c) Baboon image computed using Algorithm1 at threshold=400 , PSNR=29.5 dB and 2.8 bpp.

(d) Baboon image computed using Algorithm2 at threshold=2 , PSNR=30.3 dB and 3 bpp.

Figure 3.19: The Lena and baboon images computed using Algorithm1 and Algorithm2 architectures with a 4×4-pixel block size and 32 quantisation levels.



(a) Lena image computed using Model1 at threshold=400 , PSNR=31.5 dB and bit rate =0.9 bpp.

(b) Lena image computed using Model2 at threshold=2 , PSNR=31.7 dB and bit rate =0.9 bpp.

(c) Baboon image computed using Model1 at threshold=400 , PSNR=28.8 dB and bit rate =2.8 bpp.

(d) Baboon image computed using Model2 at threshold=2 , PSNR=29.9 dB and bit rate =3 bpp.

Figure 3.20: The Lena and baboon images computed using Model1 and Model2 architectures with a 4×4-pixel block size and 32 quantisation levels.

3.6 Summary

This chapter has introduced two low complexity image encoding algorithms based on the AQC algorithm for low bit rate applications. The two algorithms classify each image block into a low or high intensity variation. Thus, they introduce a classification threshold parameter which is used to distinguish and classify the blocks and selects the appropriate parameters to encode them. The first algorithm compares the block variance of each input block to a selected threshold to specify whether such a block is a high or low intensity variation. Such a classification operation in the second algorithm is carried out using the quantisation step. Thus, the high intensity variation blocks are encoded using a minimum, a quantisation step, a bit plane and a single bit classifier, whereas the other blocks are encoded using their mean and the classifier bit. This approach produces a low bit rate whilst preserving the PSNR and image quality within an acceptable range. The performance evaluation shows that the output bit rate and PSNR are almost the same for both algorithms. However, the second algorithm compares the quantisation step to the threshold to avoid the complexity that arises from the block variance computation. Further, the performance of both algorithms outperforms the AQC algorithm for low bit rate purposes.

In addition to the above, new efficient architectures are suggested and validated for the AQC and the proposed algorithms. The suggested architectures were implemented on a Xilinx Virtex 5 device and tested on different images, block sizes, quantisation levels and thresholds. The obtained results have shown that a speed of up to 315 MHz can be achieved. Furthermore, in the second architecture, a power consumption of 6 mW at 30 ns can be obtained with a low hardware utilisation rate, which can be used in low power consumption applications. Moreover, the hardware usage, power consumption and operating frequency of Model2 and AQC architectures are almost the same. However, Model2 architecture has an advantage in its good low bit rate-PSNR performance.

Further, the proposed algorithms and architectures in this chapter are a type of 2-D block-based low complexity systems, while with the 3-D transforms the matter is more complicated which will be introduced and discussed in chapters 4, 5 and 6.

Chapter 4: High Speed Multidimensional DCT Architectures

4.1 Introduction

The preceding chapter has presented efficient architectures for low complexity non-transform-based image compression systems. The main advantages of the proposed systems are their high speed, low complexity and low power consumption. Nevertheless, the transform-based compression systems represent an efficient alternative for high performance compression systems. However, the computation complexity of such transforms as the DCT and DWT represents the bottleneck of the entire systems. Moreover, the problem becomes much worse with multidimensional approaches, such as DCT-based image and video compression systems. This has led to the development of efficient fast algorithms and architectures for such transforms.

This chapter presents two new high speed architectures for multidimensional DCT computation. The proposed architectures are: high speed implementation of the Row Column Frame (RCF) using a 1-D DCT Radix-2 and 3-D DCT VR algorithms. The proposed architectures process the input data using an N×N×N-pixel data cube. Such architectures are tested using 8×8×8-pixel data cubes, various wordlengths and clock frequencies. The proposed architectures strike a balance between hardware resources, speed and power consumption, as can be seen from the results obtained using Virtex5 5vlx50tff1136-3 Xilinx FPGA device. The performance evaluation process has revealed that both architectures are outperforming similar architectures in terms of speed and hardware cost. Furthermore, the computation process of both architectures is carried out with a negligible error rate in a comparison with Matlab implementation.

The remainder of the chapter is organised as follows: Section 4.2 provides a description of the fast DCT algorithms. Sections 4.3 and 4.4 present two architectures for 3-D DCT computation. The results are discussed in section 4.5 and a summary is given section 4.6.

4.2 Fast Algorithms for DCT Computation

4.2.1 The 1-D DCT Radix-2 Algorithm

The computational complexity of 1-D DCT is reduced using the Radix-2 algorithm [55]. The 1-D DCT fast algorithm requires $\frac{1}{2}Nlog_2N$ multiplication and $\frac{3}{2}Nlog_2N - N + 1$ addition operations to perform N-point 1-D DCT computation. In this way, there are fewer arithmetic operations than those required by the original DCT algorithm, as illustrated in Table 2.1 [54, 55]. The fast 1-D DCT Radix-2 algorithm consists of four stages: 1-D reordering, butterfly computation, 1-D bit reverse ordering and the post addition stage, as illustrated in the block diagram of Figure 4.1.

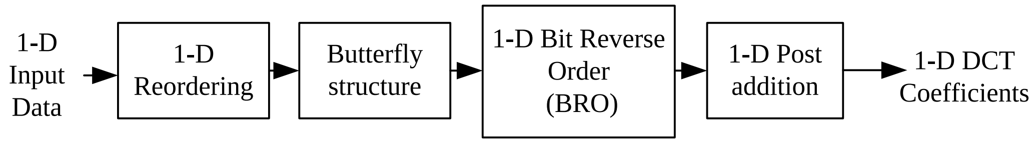


Figure 4.1: Block diagram of 1-D DCT Radix-2 algorithm.

The first stage is a reordering operation. The reordering operation of the N-point input data sequence can be expressed as follows:

$$x(n+1) = x_0(2n+1) \text{ and } x(N-n) = x_0(2n+2) \tag{4.1}$$

where x_0 and x are the input and reordered data sequence, respectively. The index n is in the range of 0 to $\frac{N}{2} - 1$.

In (4.1), the input is rearranged and fed into three butterfly sub-stages unit, which represents the main part of the algorithm. The butterfly for $N = 8$ can be calculated using the following procedure:

$$X(n+m) = x(n+m) + x(n+m+k/2)$$

$$X(n+m+k/2) = \big(x(n+m) - x(n+m+k/2)\big)cos(\emptyset)$$

$$for\ \ k = \{8,4,2\}, m = 1, k+1, k+3, k+5, \dots, N-k+1 \quad and$$

$$n = 0,1,\dots,k/2-1 \tag{4.2}$$

where $\emptyset = \frac{\pi(4n+1)}{2k}$ and each value of k represents a single butterfly sub-stage.

The next stages are the 1-D bit reverse ordering and post addition stages. The post addition for the bit-reversed ordered of eight samples can be calculated as follows:

$$Y(n) = X(n)$$

$$Y(n+1) = X(n+1)$$

$$Y(n+2) = X(n+2)$$

$$Y(n+4) = X(n+4)$$

$$Y(n+3) = 2X(n+3) - X(n+2)$$

$$Y(n+6) = 2X(n+6) - X(n+4)$$

$$Y(n+5) = 2X(n+5) - Y(n+6)$$

$$Y(n+7) = 4X(n+7) - 2X(n+6) - Y(n+5) \tag{4.3}$$

where $n = 0, 8, 16, 24 \; \ldots., P - 7$, P is the input data size, X is the output of the butterfly stage and Y is the output of the post addition stage. The final 1-D DCT coefficients requires normalisation; dividing each element by two (shift right operation), with the exception of the first element (DC element) which has to be divided by $(2\sqrt{2})$.

Further, the 3-D DCT Radix-2 RCF technique is performed by applying a 1-D DCT Radix-2 on the rows, columns and frames, successively. This approach requires $\frac{3}{2}N^3 log_2 N$ multiplication and $\frac{9}{2}N^3 log_2 N - 3N^3 + 3N^2$ addition operations [54].

4.2.2 The 3-D DCT VR Algorithm

Although the computation of the 3-D DCT RCF using 1-D DCT Radix-2 algorithm requires fewer multiplication operations than the direct 3-D DCT, further computation load reduction can be achieved [54, 56, 57]. Hence, an efficient algorithm for fast 3-D DCT computation is the 3-D DCT VR algorithm, which was introduced in [54, 56]. This algorithm requires $\frac{7}{8}N^3 log_2 N$ multiplication and $\frac{9}{2}N^3 log_2 N - 3N^3 + 3N^2$ addition operations for $N \times N \times N$-point 3-D DCT computation. This algorithm computes the 3-D DCT for specific data cubes according to the following sequence:

- 3-D reordering
- Butterfly calculation
- 3-D bit reverse order

- 3-D post addition

This algorithm assumes that the input data is a $N \times N \times N$-pixel, where N is a power of two. In this algorithm, the 3-D input data is first rearranged according to the following index mapping:

$$\begin{bmatrix} \tilde{x}(n_1, n_2, n_3) \\ \tilde{x}(n_1, n_2, N - n_3 - 1) \\ \tilde{x}(n_1, N - n_2 - 1, n_3) \\ \tilde{x}(n_1, N - n_2 - 1, N - n_3 - 1) \\ \tilde{x}(N - n_1 - 1, n_2, n_3) \\ \tilde{x}(N - n_1 - 1, n_2, N - n_3 - 1) \\ \tilde{x}(N - n_1 - 1, N - n_2 - 1, n_3) \\ \tilde{x}(N - n_1 - 1, N - n_2 - 1, N - n_3 - 1) \end{bmatrix} = \begin{bmatrix} x(2n_1, 2n_2, 2n_3) \\ x(2n_1, 2n_2, 2n_3 + 1) \\ x(2n_1, 2n_2 + 1, 2n_3) \\ x(2n_1, 2n_2 + 1, 2n_3 + 1) \\ x(2n_1 + 1, n_2, n_3) \\ x(2n_1 + 1, n_2, 2n_3 + 1) \\ x(2n_1 + 1, 2n_2 + 1, n_3) \\ x(2n_1 + 1, 2n_2 + 1, 2n_3 + 1) \end{bmatrix} \quad (4.4)$$

where $n_i = 0, 1, \ldots, \frac{N}{2} - 1$, i=1,2,3, and $\tilde{x}$ is the reordered form of the original input x.

Equation (2.5) can be rewritten by exchanging $x(n_1, n_2, n_3)$ with $\tilde{x}(n_1, n_2, n_3)$:

$$X(k_1, k_2, k_3) = \frac{8}{N^3} \varepsilon_{k_1} \varepsilon_{k_2} \varepsilon_{k_3} \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} \sum_{n_3=0}^{N-1} \tilde{x}(n_1, n_2, n_3) \times cos(\emptyset_1 k_1)$$

$$\times cos(\emptyset_2 k_2) \times cos(\emptyset_3 k_3) \quad (4.5)$$

where $\emptyset_i = \frac{\pi}{2N}(4n_i + 1)$ and *i=1,2,3.*

The computation of the 3-D DCT VR algorithm consists of two steps: butterflies and post addition stages. The number of stages requires for the butterfly computation is $log_2 N$ with $\frac{N^3}{8}$ butterfly per stage. The multiplication by the normalisation factor $\frac{8}{N^3} \varepsilon_{k_1} \varepsilon_{k_2} \varepsilon_{k_3}$, can be postponed to the final stage.

By considering even and odd parts, the 3-D DCT can be computed as follows:

$$X(2k_1 + i, 2k_2 + j, 2k_3 + l) = \sum_{n_1=0}^{M} \sum_{n_2=0}^{M} \sum_{n_3=0}^{M} \left[\tilde{x}_{ijl}(n_1, n_2, n_3)\right]$$

$$\times cos\big(\emptyset_1(k_1 + i)\big) \times cos\big(\emptyset_2(k_1 + j)\big) \times cos\big(\emptyset_3(k_1 + l)\big) \quad (4.6)$$

where $M = {}^{N}\!/_{2} - 1$ and *ijl*=[000,001,010,011,100,101,110,111] and $\tilde{x}_{ijl}(n_1, n_2, n_3)$ is expressed as:

$$\begin{aligned}\tilde{x}_{ijl}&(n_1,n_2,n_3)\\ &= \tilde{x}(n_1,n_2,n_3) + (-1)^l\tilde{x}\left(n_1,n_2,n_3+\frac{n}{2}\right)\\ &+ (-1)^j\tilde{x}\left(n_1,n_2+\frac{n}{2},n_3\right) + (-1)^{j+l}\tilde{x}\left(n_1,n_2+\frac{n}{2},n_3+\frac{n}{2}\right)\\ &+ (-1)^i\tilde{x}\left(n_1+\frac{n}{2},n_2,n_3\right) + (-1)^{i+l}\tilde{x}\left(n_1+\frac{n}{2},n_2,n_3+\frac{n}{2}\right)\\ &+ (-1)^{i+j}\tilde{x}\left(n_1+\frac{n}{2},n_2+\frac{n}{2},n_3\right)\\ &+ (-1)^{i+j+l}\tilde{x}\left(n_1+\frac{n}{2},n_2+\frac{n}{2},n_3+\frac{n}{2}\right)\end{aligned} \tag{4.7}$$

For even and odd values of k_1, k_2 and k_3, (4.6) can be developed as follows:

$$X(2k_1,2k_2,2k_3) = \left\{\sum_{n_1=0}^{M}\sum_{n_2=0}^{M}\sum_{n_3=0}^{M}\ [\tilde{x}_{000}(n_1,n_2,n_3)] \times \prod_{i=1}^{3} cos(2\emptyset_i k_i)\right\} \tag{4.8}$$

where $M = {}^{N}\!/_{2} - 1$.

$$\begin{aligned}X(2k_1,&2k_2,2k_3+1)\\ &= \left\{\sum_{n_1=0}^{M}\sum_{n_2=0}^{M}\sum_{n_3=0}^{M}[2\tilde{x}_{001}(n_1,n_2,n_3)\cos\emptyset_3] \times \prod_{i=1}^{3} cos(2\emptyset_i k_i)\right\}\\ &- X(2k_1,2k_2,2k_3-1)\end{aligned} \tag{4.9}$$

$$\begin{aligned}X(2k_1,&2k_2+1,2k_3)\\ &= \left\{\sum_{n_1=0}^{M}\sum_{n_2=0}^{M}\sum_{n_3=0}^{M}[2\tilde{x}_{010}(n_1,n_2,n_3)\cos\emptyset_2] \times \prod_{i=1}^{3} cos(2\emptyset_i k_i)\right\}\\ &- X(2k_1,2k_2-1,2k_3)\end{aligned} \tag{4.10}$$

$$\begin{aligned}X(2k_1&+1,2k_2,2k_3)\\ &= \left\{\sum_{n_1=0}^{M}\sum_{n_2=0}^{M}\sum_{n_3=0}^{M}[2\tilde{x}_{100}(n_1,n_2,n_3)\cos\emptyset_1] \times \prod_{i=1}^{3} cos(2\emptyset_i k_i)\right\}\\ &- X(2k_1-1,2k_2,2k_3)\end{aligned} \tag{4.11}$$

$$X(2k_1, 2k_2+1, 2k_3+1)$$

$$= \left\{ \sum_{n_1=0}^{M} \sum_{n_2=0}^{M} \sum_{n_3=0}^{M} [4\tilde{x}_{011}(n_1, n_2, n_3) \cos \emptyset_2 \cos \emptyset_3] \times \prod_{i=1}^{3} cos(2\emptyset_i k_i) \right\}$$

$$-X(2k_1, 2k_2-1, 2k_3+1)$$

$$-X(2k_1, 2k_2+1, 2k_3-1)$$

$$-X(2k_1, 2k_2-1, 2k_3-1) \tag{4.12}$$

$$X(2k_1+1, 2k_2, 2k_3+1)$$

$$= \left\{ \sum_{n_1=0}^{M} \sum_{n_2=0}^{M} \sum_{n_3=0}^{M} [4\tilde{x}_{101}(n_1, n_2, n_3) \cos \emptyset_1 \cos \emptyset_3] \times \prod_{i=1}^{3} cos(2\emptyset_i k_i) \right\} - X(2k_1-1, 2k_2, 2k_3+1)$$

$$- X(2k_1+1, 2k_2, 2k_3-1) - X(2k_1-1, 2k_2, 2k_3-1) \tag{4.13}$$

$$X(2k_1+1, 2k_2+1, 2k_3)$$

$$= \left\{ \sum_{n_1=0}^{M} \sum_{n_2=0}^{M} \sum_{n_3=0}^{M} [4\tilde{x}_{110}(n_1, n_2, n_3) \cos \emptyset_1 \cos \emptyset_2] \times \prod_{i=1}^{3} cos(2\emptyset_i k_i) \right\}$$

$$- X(2k_1-1, 2k_2+1, 2k_3) - X(2k_1+1, 2k_2-1, 2k_3)$$

$$- X(2k_1-1, 2k_2-1, 2k_3) \tag{4.14}$$

$$X(2k_1+1, 2k_2+1, 2k_3+1)$$

$$= \left\{ \sum_{n_1=0}^{M} \sum_{n_2=0}^{M} \sum_{n_3=0}^{M} [8\tilde{x}_{111}(n_1, n_2, n_3) \cos \emptyset_1 \cos \emptyset_2 \cos \emptyset_3] \times \prod_{i=1}^{3} cos(2\emptyset_i k_i) \right\}$$

$$- X(2k_1+1, 2k_2+1, 2k_3-1) - X(2k_1+1, 2k_2-1, 2k_3+1)$$

$$- X(2k_1+1, 2k_2-1, 2k_3-1) - X(2k_1-1, 2k_2+1, 2k_3+1)$$

$$- X(2k_1-1, 2k_2+1, 2k_3-1) - X(2k_1-1, 2k_2-1, 2k_3+1)$$

$$- X(2k_1-1, 2k_2-1, 2k_3-1) \tag{4.15}$$

Equations (4.8)-(4.15) contain two parts: the first is the term between the curly brackets for butterfly computation and the second represents the computation operations carried out in the post addition stage.

4.3 3-D DCT RCF Architecture (RCF Architecture)

The proposed RCF Architecture computes an N-point 1-D DCT Radix-2 algorithm on the rows, columns and frames in a $N \times N \times N$-point data cube. Thus, this architecture is composed of three N-point 1-D DCT stages and data control units, as illustrated in Figure 4.2. The computation sequence of this architecture is as follows:

- 1-D DCT on the rows.
- Normalisation
- Transpose buffer1 (TMem1)
- 1-D DCT on the columns.
- Normalisation
- Transpose buffer2 (TMem2)
- 1-D DCT on the frames.
- Normalisation
- Reordering buffer (TMem3)

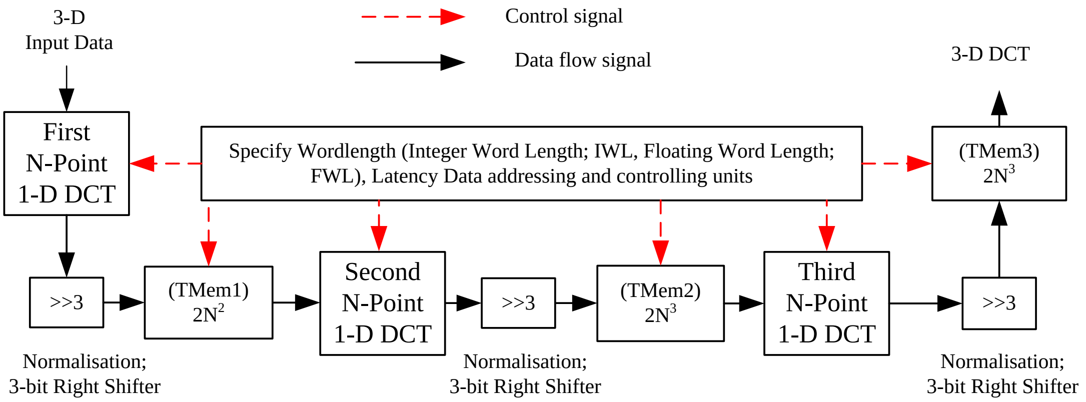


Figure 4.2: Block diagram of the whole RCF architecture.

In Figure 4.2, the dashed red arrows represent the controller signals and the continuous arrows represent the data flow signals. The wordlength and latency are specified for each stage prior to the computation process using input parameters in the controller unit. Moreover, the architecture of each 1-D DCT stage is composed of serial to parallel converter, butterfly, bit reverse ordering, post addition and multiplexer unit, as illustrated in Figure 4.3.

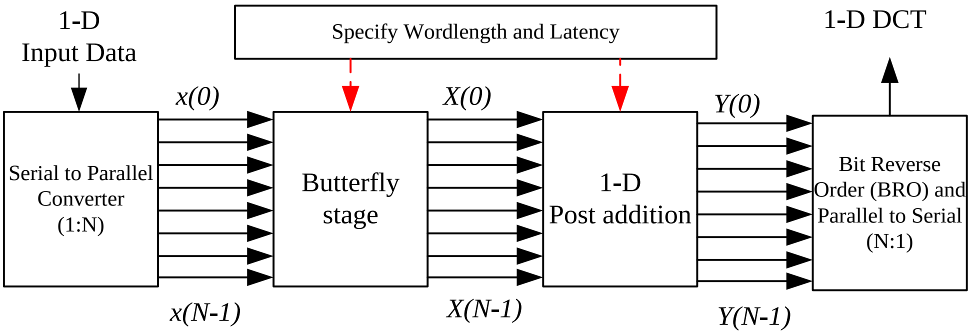

1-D
Input Data
Specify Wordlength and Latency
1-D DCT
x(0)
X(0)
Y(0)
Serial to Parallel Converter (1:N)
Butterfly stage
1-D Post addition
Bit Reverse Order (BRO) and Parallel to Serial (N:1)
x(N-1)
X(N-1)
Y(N-1)

Figure 4.3: Block diagram of N-point 1-D DCT architecture.

The input data to the first 1-D DCT stage is converted to a group of N-parallel pixels using the serial to parallel converter and fed to the butterfly stage. The butterfly stage performs addition, subtraction and twiddle factor multiplication operations. It is composed of three arithmetic operations sub-stages with a varying latency and wordlength, as illustrated in Figure 4.4. The latency can be altered easily without affecting the whole operation of the proposed architecture, which can improve the speed and reduce the power consumption. However, such improvements come at an extra hardware cost. The latency can be adjusted using three input parameters: $A(i,j)$, $S(i,j)$ and $M(i,j)$ for adders, subtracters and multipliers, respectively. Where i represents the sub-stage in each butterfly (i=*1,2,3*) and j is the butterfly stage number; j =*1,2* and *3;* the first, second and third 1-D DCT stages, respectively.

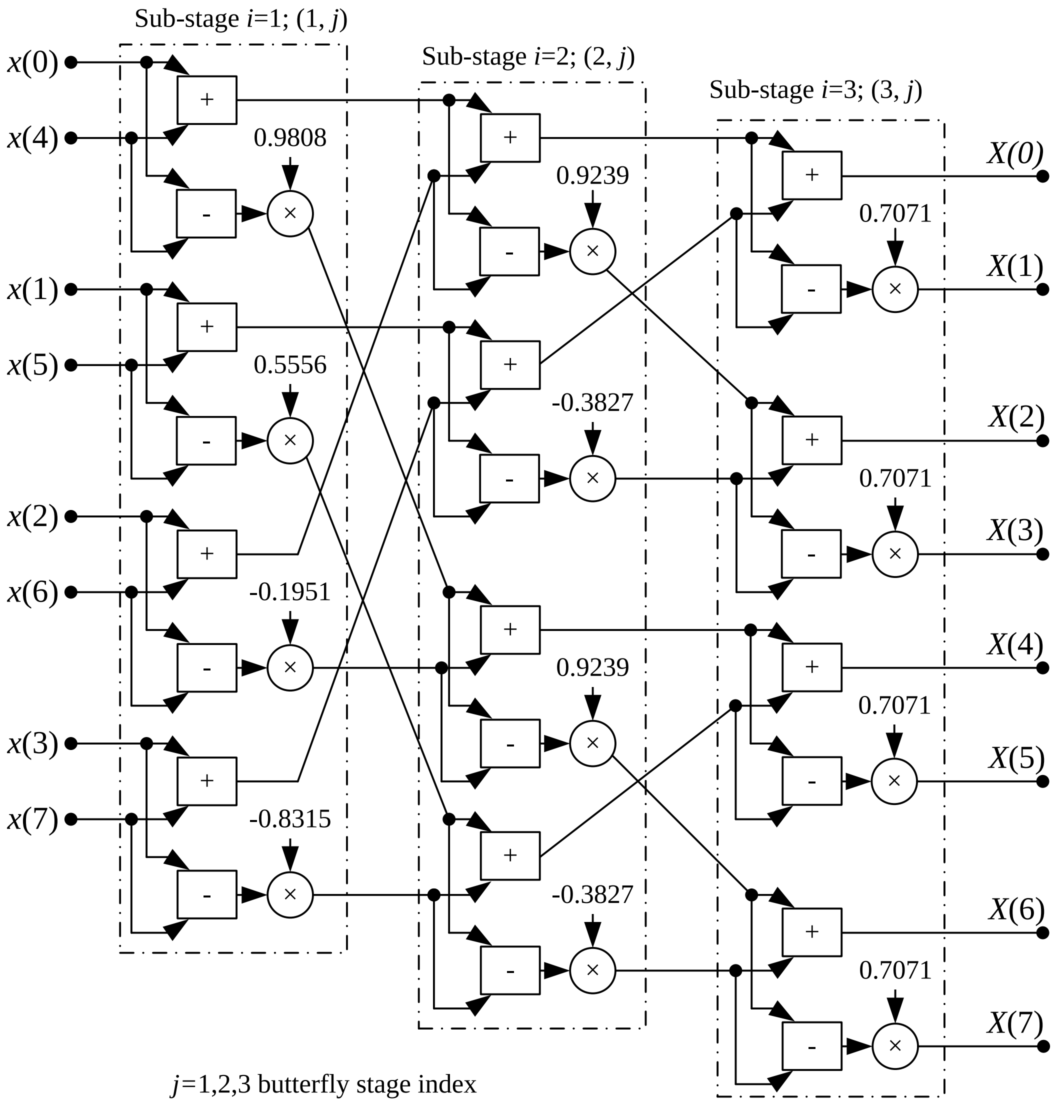


Figure 4.4: Block diagram of 8-point butterfly stage (three sub-stages).

The output precision can be controlled by modifying the wordlength of each unit. Two designed parameters have been used to specify the wordlength: the Integer Wordlength (IWL) and Floating Wordlength (FWL). The IWL represents the number of bits required for the signed integer part and FWL represents the number of bits for the mantissa part.

The subsequent sub-stage is the post-addition which is implemented using bit shifters and subtracters, as illustrated in Figure 4.5. The post addition stage performs the addition sequence that is described in (4.3). Additional registers are added for pipelining and synchronising purposes. When the preceding data processing sequence is completed, the parallel data is converted into a serial stream and normalised by a factor of ${}^{1}/_{N}$.

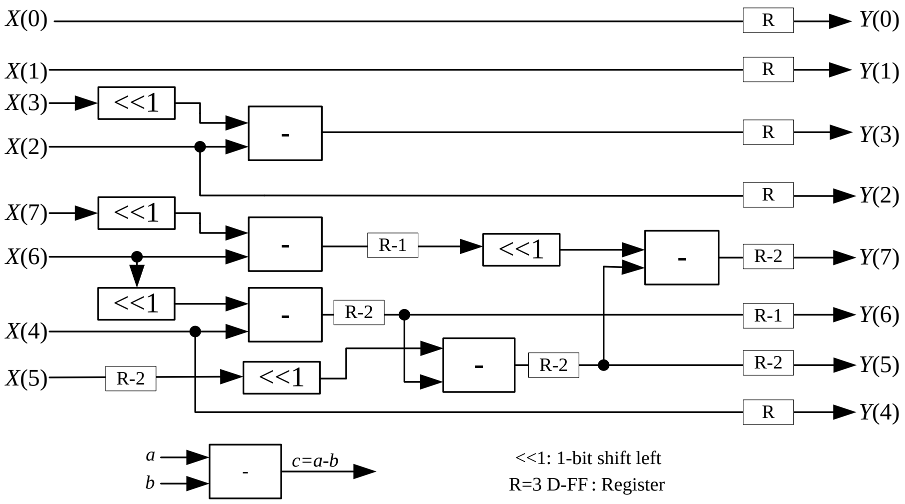


Figure 4.5: Block diagram of 8-point post addition stage.

The output data from the first 1-D DCT will be available after d_{MW} clock cycles, where the period d_{MW} can be computed as follows:

$$d_{MW} = N \times 3L(i,j) + MuxD_1 + PAdd_{delay} + \psi \tag{4.16}$$

where $L(i,j)$ is the latency in each sub-stage (i=1, 2, 3) for the first butterfly stage when j=1, $MuxD_1$ is the latency of the parallel to the serial convertor, $PAdd_{delay}$ represents the latency of the post-addition stage, and ψ represents any other latency offset. The normalisation operation is implemented using a log_2N-bit right shift register to avoid any overflow in the subsequent arithmetic operations. The normalised output is then written into a N^2-word Block RAM; TMem1. The memory TMem1 is activated by the control signal W_enb1 for the data to be written at addresses between 0 to $N^2 - 1$ that is carried by the control signal W_adds1, as shown in Figure 4.6.

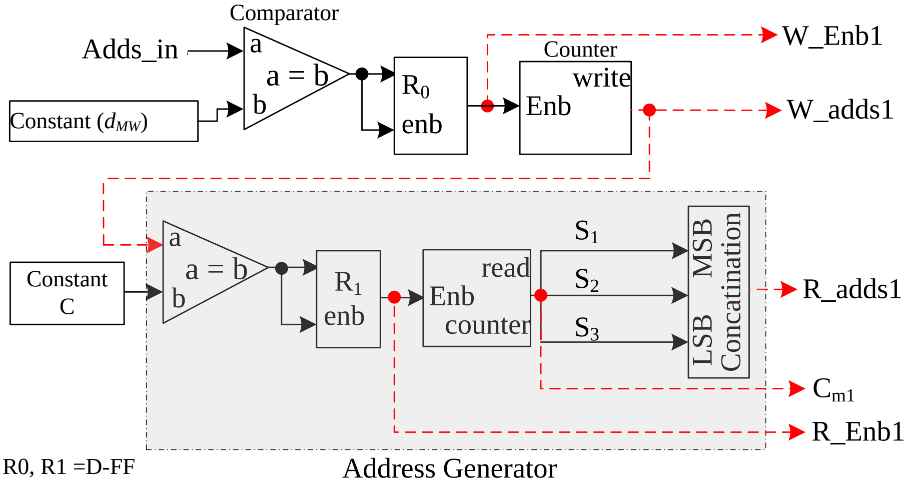


Figure 4.6: TMem1 writing /reading control unit of the first 1-D DCT stage.

Data stored in TMem1 is reordered and fed to the second 1-D DCT stage. The reordering operation is carried out using R_Adds1, as shown in Figure 4.6. The value carried by R_adds1 signal is given as:

$$\text{R\_adds1} = l + mN \text{ , } for \; 0 \le l \le N-1 \; and \; 0 \le m \le N-1 \qquad (4.17)$$

where l and m represent the rows and the columns indices of an $N \times N$-point data block, respectively. The R_Adds1 values are generated using the shaded part of Figure 4.6 and Table 4.1, and it can be represented as follows:

$$S_3 = X_n \; for \; 1 \le n \le \log_2 \mathrm{N}$$

$$S_2 = X_n \; for \; \log_2 \mathrm{N} < n \le 2\log_2 \mathrm{N}$$

$$S_1 = X_n \; for \; n > 2\log_2 \mathrm{N}$$

$$\text{then,} \qquad \text{R\_adds1} = [S_1 \; S_3 \; S_2] \qquad (4.18)$$

where X_n is the binary representation of a $log_2 2N^2$-bit counter, $n = 1,2, \ldots log_2 2N^2$, $n = 1$ is the least significant bit (LSB) and the acronym S refers to bit slice.

From Figure 4.6, the outputs of this control circuit are: TMem1 writing enables (W_enb1), writing address (W_adds1), reading address (R_adds1), reading enables (R_enb1) and counter ($\mathrm{C_{m1}}$). The $\mathrm{C_{m1}}$ signal is used to control the multiplexer of the subsequent 1-D DCT stage.

Table 4.1: The first eight addresses (R_adds1) for 8×8-point block.

Main counter (Input addresses-Adds_in)								Output addresses (R_adds1)							
Decimal	Binary							Binary							Decimal
	S_1	S_2			S_3			S_1	S_3			S_2			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	8
2	0	0	0	0	0	1	0	0	0	1	0	0	0	0	16
3	0	0	0	0	0	1	1	0	0	1	1	0	0	0	24
4	0	0	0	0	1	0	0	0	1	0	0	0	0	0	32
5	0	0	0	0	1	0	1	0	1	0	1	0	0	0	40
6	0	0	0	0	1	1	0	0	1	1	0	0	0	0	48
7	0	0	0	0	1	1	1	0	1	1	1	0	0	0	56

The second 1-D DCT stage performs the same operations sequence of the second 1-D DCT on the columns of each $N \times N$-point data block. The output is stored in a $2N^3$-word block memory; TMem2, as shown in Figure 4.2. The TMem2 writing address, W_adds2 signal is generated by a $log_2 2N^3$-bit counter and bit slices as shown in Figure 4.7-a. The address generator in Figure 4.7-a is constructed in the same way as that shown in the shaded part of Figure 4.6 with different bit slices, which can be computed as follows:

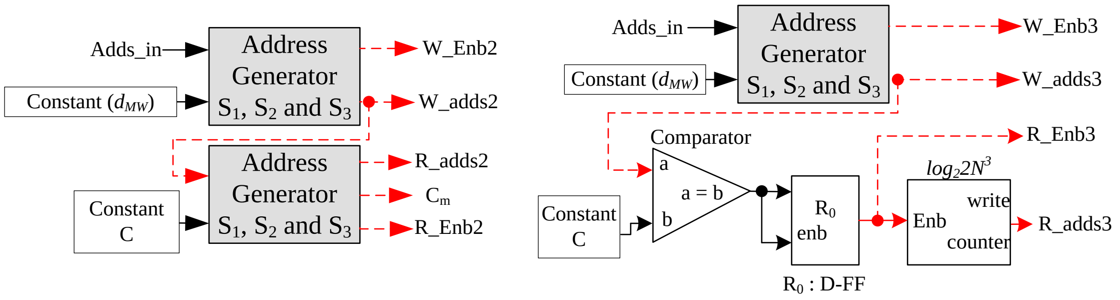
$$S_3 = X_n \; for \; 1 \le n \le \log_2 \mathrm{N}$$

$$S_2 = X_n \; for \; \log_2 \mathrm{N} < n \le 2 \log_2 \mathrm{N}$$

$$S_1 = X_n \; for \; n > 2 \log_2 \mathrm{N}$$

$$\text{then,} \qquad \text{W\_adds2} = [S_1 \; S_3 \; S_2] \qquad (4.19)$$

where X_n is the binary representation of a $log_2 2N^3$-bit counter.



a. Block diagram of the second controller.

b. Block diagram of the third controller.

Figure 4.7: Block diagrams of the second and third memory writing/reading control units (for TMem2 and TMem3).

The data stored in TMem2 is reordered and fed to the third 1-D DCT stage to perform a transform along the frame direction. TMem2 reordering operation is carried out as shown in Figure 4.7-a, and it can be expressed as follows:

$$S_3 = X_n \; for \; 1 \le n \le log_2 N$$

$$S_2 = X_n \; for \; log_2 N < n \le 3 \, log_2 N$$

$$S_1 = X_n \; for \; n > 3 \, log_2 N$$

$$\text{then,} \qquad \text{R\_adds2} = [S_1 \; S_3 \; S_2] \qquad (4.20)$$

Hence, the 3-D DCT of the input data is computed. The computed results are stored in the third block RAM; TMem3, as shown in Figure 4.2. TMem3 acts as a reordering buffer to produce an output sequence with the same indexing order as the original data. TMem3 writing address signal; W_adds3, is generated similar to R_adds2, while its reading address; R_adds3, is generated using a $log_2 2N^3$-bit counter only, as shown in Figure 4.7-b.

4.4 The Proposed 3-D DCT VR Architecture

The proposed 3-D DCT VR architecture is shown in Figure 4.8 and it consists of the following:

- 3-D reordering
- Three butterfly stages
- 3-D bit reverse ordering
- Three post addition stages
- Normalisation

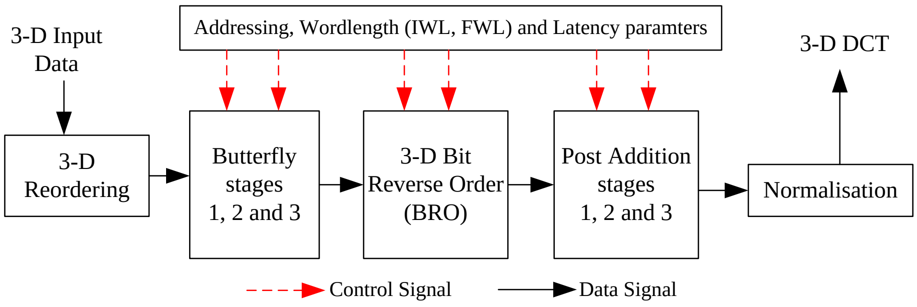

3-D Input Data
Addressing, Wordlength (IWL, FWL) and Latency paramters
3-D DCT
3-D Reordering
Butterfly stages 1, 2 and 3
3-D Bit Reverse Order (BRO)
Post Addition stages 1, 2 and 3
Normalisation
Control Signal
Data Signal

Figure 4.8: Block diagram of 3-D DCT VR architecture.

The timing sequence of the proposed architecture is shown in Figure 4.9. In this figure, each block represents N^3 points, and the delay ($\boldsymbol{\delta}_1$, $\boldsymbol{\delta}_2$ ... $\boldsymbol{\delta}_6$) between each of the consecutive stages is selected according to the availability of the data required by the ensuing stage. Furthermore, two parameters are used to control the wordlength of the proposed architecture stages termed; IWL and FWL.

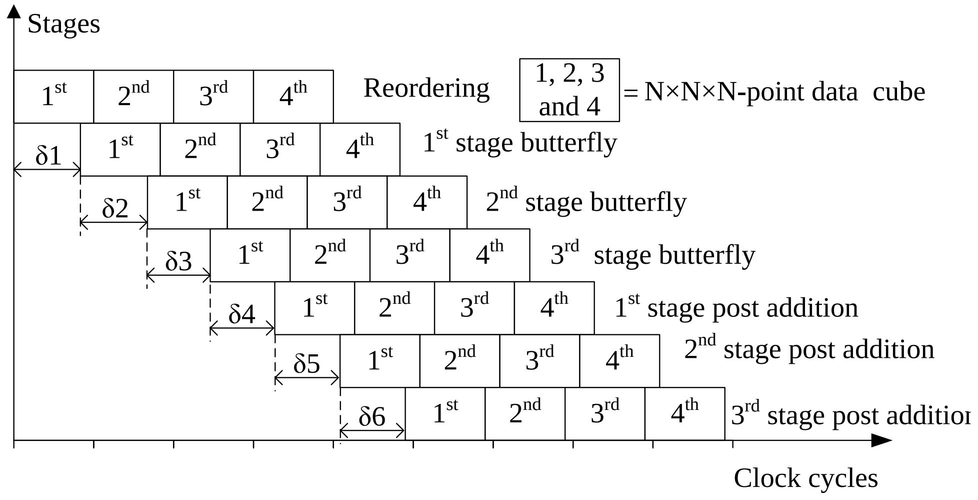


Figure 4.9: The timing sequence of the proposed 3-D DCT VR architecture.

4.4.1 The 3-D Reordering Stage

The reordering stage consists of counters, ROMs and block memory (TMem1), as shown in Figure 4.10. Three signals termed; Data_out, C_m and R_adds are generated by the reordering stage. The first signal; Data_out, represents the output data stream and the second signal; C_m is a $log_2 2N^3$-bit counter. Signal C_m is used to control the parallel to serial converters and to select the appropriate twiddle factors of the butterfly stages. The third signal; R_adds, is used to select the proper data for the first butterfly stage. The reordering operation is carried out as shown the following two subdivisions.

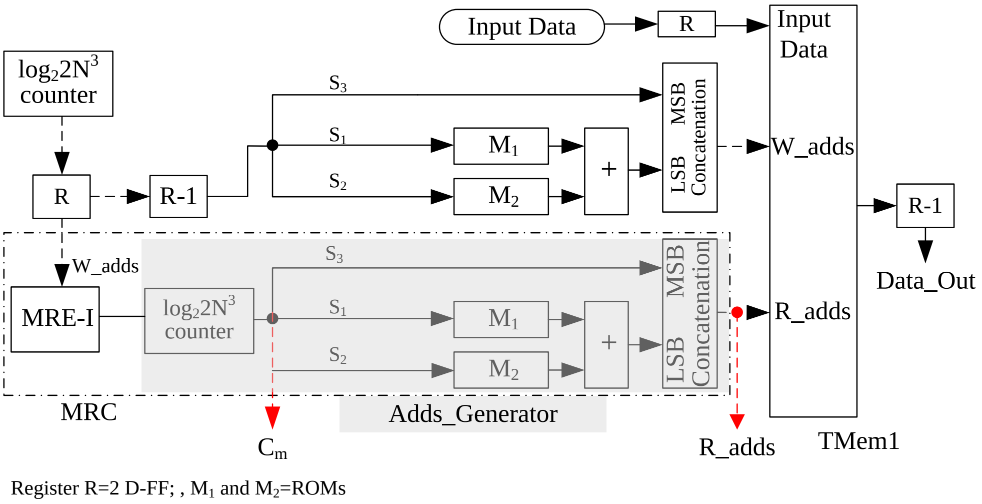


Figure 4.10: Block diagram of the reordering unit.

4.4.1.1 Memory Writing Operation

The reordering operation is carried out during the memory writing operation to TMem1. The writing addresses are generated using two ROMs; M_1 of N^2-word and M_2 of N-word and their addressing units. The memory writing addresses; reordering addresses are computed according to:

$$\mathrm{W\_adds} = \left[\mathrm{S}_3 \quad \left(\mathrm{M}_1(\mathrm{S}_1) + \mathrm{M}_2(\mathrm{S}_2)\right)\right] \tag{4.21}$$

where the annotation [ ] indicates the concatenation operation of the content between the brackets, M_1 and M_2 represent two ROMs, $\mathrm{S}_1 = 0,1,2,3\ldots$, $(\mathrm{N}^2 - 1)$, $\mathrm{S}_2 = 0,1,2,\ldots N - 1$ and the clock period of S_1 is N times of that of S_2. Signal S_3 is the MSB of the $\log_2 2N^3$-bit counter C_m. Signals S_1, S_2 and S_3 can be expressed as follows:

$$S_1 = C_m\, for\ \ log_2 N < n < 3\, log_2 N$$

$$S_2 = C_m\, for\ 1 \le n \le log_2 N$$

$$S_3 = C_m\, for\ n = 3\, log_2 N \tag{4.22}$$

The actual content of M_2 is shown in the first column of Table 4.2, while the content of M_1 shown in columns 2 to 9 of the same table. As an example of data reordering for *N=8*, when S_1, S_3=0 and S_2=0, 1, 2, 3,..7, the TMem1 writing addresses of the first eight points are: 0, 388, 4, 384, 128, 260, 132 and 256. Similarly, the addresses of the second eight points are: 98, 486, 102, 482, 226, 358, 230 and 354. In other words, each element in ROM M_1 is added to the content of ROM M_2 in order to generate the memory writing addresses.

Table 4.2: Content of ROMs M_1 and M_2 for the reordering operation (TMem1 Writing Operation) [(1)].

M_2 Content	M_1 Content							
0	0	98	2	96	32	66	34	64
388	25	123	27	121	57	91	59	89
4	1	99	3	97	33	67	35	65
384	24	122	26	120	56	90	58	88
128	8	106	10	104	40	74	42	72
260	17	115	19	113	49	83	51	81
132	9	107	11	105	41	75	43	73
256	16	114	18	112	48	82	50	80

[(1)] Reading sequence of M_1 content is 0, 98, 2, 96, 32, 66, 34, 64, 25, 123 …… 80.

4.4.1.2 Memory Reading Operation

The output of memory component TMem1 represents the required input data to the ensuing butterfly stage. The start of TMem1 reading operation is controlled by the Memory Reading Controller (MRC), as shown in the shaded part of Figure 4.10. The MRC consists of two units, the Memory Reading Enable (MRE) and the Address Generator (Adds_Generator). The functions of the MRE unit are to avoid using long delay elements and control the start of memory reading operations, as shown in Figure 4.11. Two MRE components are proposed, namely; MRE-I and MRE-II, as illustrated in Figure 4.11. Component MRE-I generates the memory reading enabling signal R_enb according to the comparison between the W_adds and the parameter α. While, Component MRE-II generates R_enb by comparing accumulated W_enb with the parameter α. The parameter α specifies the time to start memory reading operation and it is shown underneath Figure 4.12 for N=8. Further, MRE-I is used in the reordering stage while the MRE-II is used in other memory reading controllers.

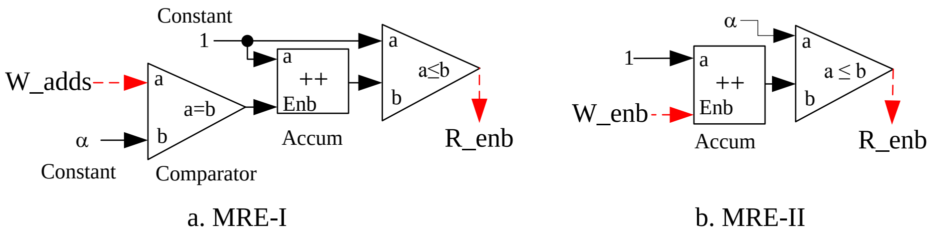
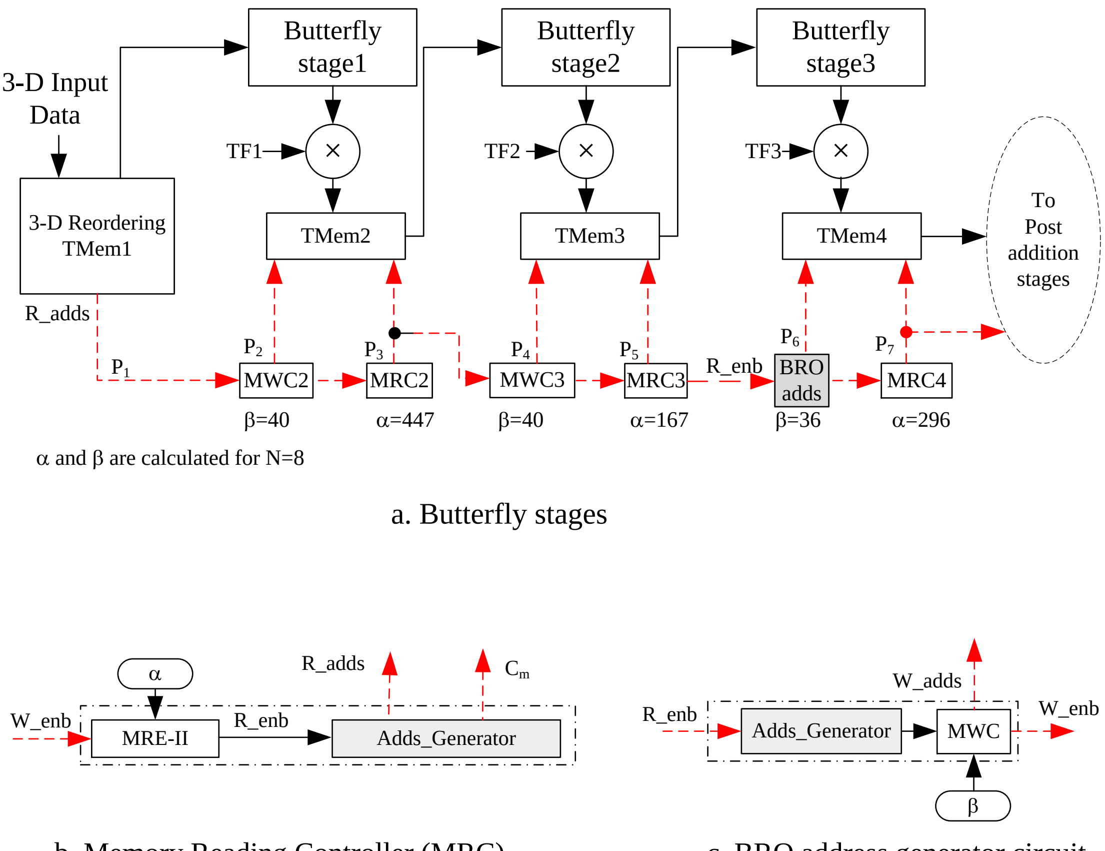


Figure 4.11: MRE circuits.

The Adds_Generator unit is used to generate the memory reading addresses R_adds in similar way to W_adds generation, though with different ROMs contents. Such contents are listed in Appendix C.

4.4.2 Butterfly Stages

The butterfly computation consists of three similar stages, as shown in Figure 4.12. The differences between these three butterfly stages are the values of twiddle factors and the memory read/write addresses. From Figure 4.12, each butterfly stage is connected to a buffer; TMem2, 3 and 4 and their corresponding memory writing and reading controllers, MWC and MRC, respectively. In this figure, TF1, 2 and 3 are the twiddle factors of each butterfly stage. Signals MWCs, MRCs are the memory writing and reading addresses controllers, respectively. The addresses at points P_1 to P_7 can be computed as shown in appendix C. In addition, β and α are controlling parameters used in MWC and MRE units, respectively. Such parameters are used to specify the start of memory writing and reading operations in each stage in the proposed architecture. The values of these parameters depend on the latency of each stage. In Figure 4.12, these parameters are calculated for N=8.



a. Butterfly stages

b. Memory Reading Controller (MRC)

c. BRO address generator circuit

Figure 4.12: Block diagrams of the butterfly stages and, the MRC and BRO address controllers.

Each butterfly stage consists of a serial to parallel converter, an 8-points butterfly structure and parallel to serial converter, as shown in Figure 4.13. From this figure, the input data is converted to N-parallel points to perform the butterfly computation as shown in Figure 4.14. The twiddle factors multiplication operation is postponed after the parallel to serial converter to reduce the number of multipliers to a single multiplier per butterfly stage.

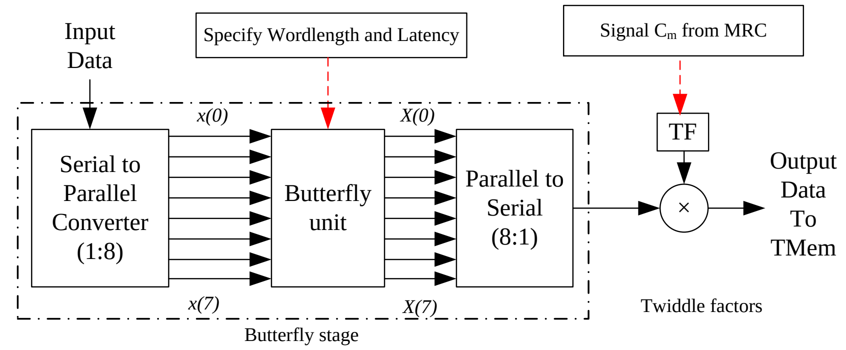


Figure 4.13: Block diagram for single butterfly stage.

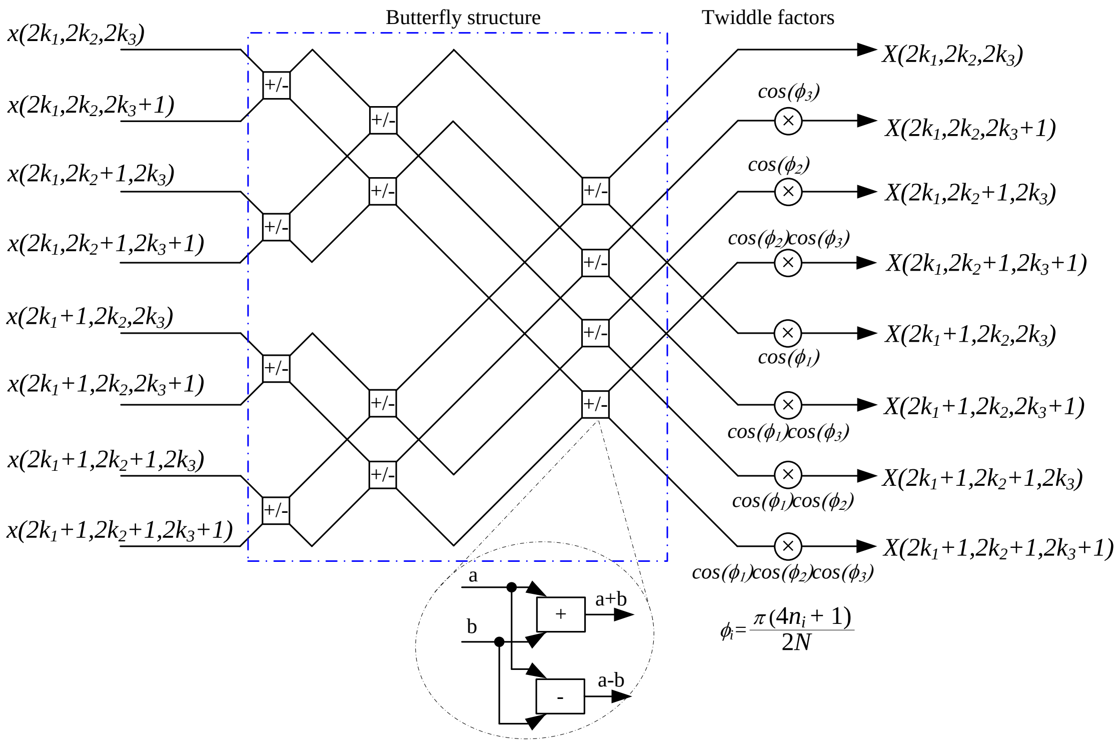


Figure 4.14: Eight-point butterfly structure.

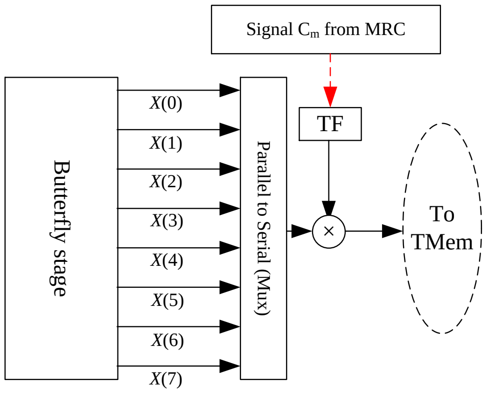


Figure 4.15: The twiddle factor multiplication circuit for single butterfly stage.

Once the butterfly computation is completed the data is converted to a serial stream and a multiplication by twiddle factors is performed, as shown in Figure 4.15. In Figure 4.15, the annotation TF refers to pre-computed twiddle factors for butterfly stages 1, 2 and 3 using three ROMS. The ROM sizes are: N^3, N^2 and N words for TF1, TF2 and, TF3, respectively. The controller signal for each ROM is generated from the corresponding MRC block; bit-slice from the signal C_m. The content of each ROM is computed according to the cosine terms in (4.8) to (4.15).

The output of the first butterfly stage is stored in the second memory TMem2 which acts as a reordering buffer for the ensuing butterfly stage. The buffer TMem2 is controlled by Memory Writing Controller (MWC) circuit, as shown in Figure 4.16. After β clock cycles, the memory enabling signal is high W_enb=1, if the addresses signals adds_1 and adds_2 are different:

$$\text{W\_enb} = \begin{cases} 1 & \text{adds}_1 \neq \text{adds}_2 \\ 0 & otherwise \end{cases} \tag{4.23}$$

where adds_2 is the adds_1 delayed by one cycle, and adds_1 is the memory reading addresses of the preceding stage delayed by β cycles. The values of β depends on the latency of each component in the butterfly stage. An important advantage of this circuit is to reduce the power consumption by turning on the memory component for writing operation at specific times.

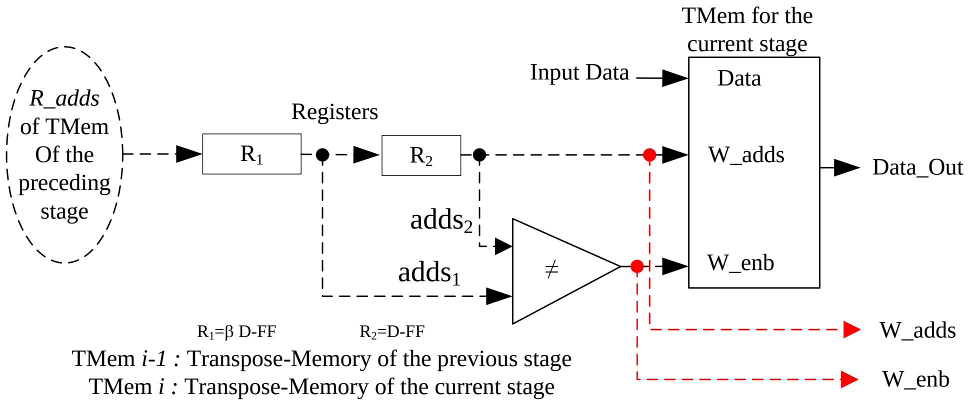


Figure 4.16: Memory Writing Controller (MWC) block diagram.

The computation sequence of the subsequent two butterfly stages is similar to the first butterfly stage. The output from the second and third butterfly stages is stored in memory components TMem3 and TMem4, respectively. Furthermore, the output from the third butterfly stage is stored in a different order, by adjusting the memory writing addresses to perform a 3-D Bit Reverse Order (BRO) operation. The BRO addresses at point P_6 in Figure 4.12 are shown in the Appendix C.

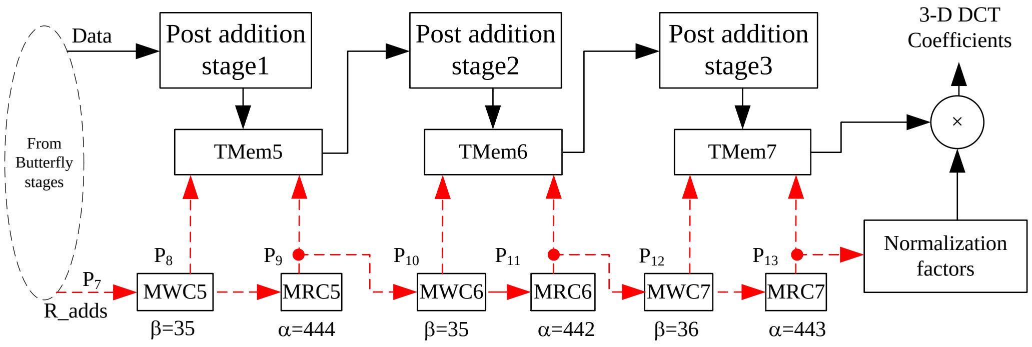
4.4.3 Post Addition Stages

Three post addition stages are employed to compute the terms that are outside the curly brackets in equations (4.8)-(4.15), as shown in Figure 4.17. The output of the three post addition stages is stored in TMem5, 6 and 7, respectively. The memory TMem7 is used as a reordering buffer to rearrange the output coefficients to the same order of the input data. The addresses at each point in Figure 4.17 for N=8 are detailed in the Appendix C. Each post addition stage is composed of serial to parallel converter, 8-point addition unit and parallel to serial converter. The 8-point addition unit of each post-addition stage consists of five subtracters and additional registers for pipelining purposes, as shown in Figure 4.18. The length of each register can be adjusted using input parameter without affecting the validity of the proposed architecture. Furthermore, the wordlength of each stage are controlled using two input parameters, IWL and FWL.

The final stage of the proposed architecture is the normalisation operation which can be specified by:

$$NormF = \frac{8}{N^3}\varepsilon_{k_1}\varepsilon_{k_2}\varepsilon_{k_3} \tag{4.24}$$

where $k_i = 0, 1, 2, \dots N-1,\ \ i = 1, 2, 3$ and $\varepsilon_{k_i} = \begin{cases} \frac{1}{\sqrt{2}}, & for\ k_i = 0 \\ 1, & otherwise \end{cases}$.



α and β are calculated for N=8

Figure 4.17: The block diagram of the post addition stages.

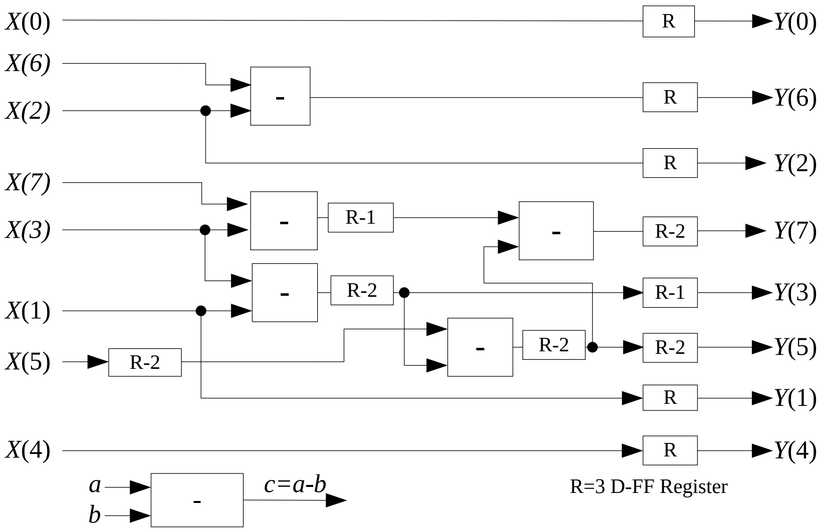


Figure 4.18: Eight points addition unit of each post addition stage.

4.5 Results and Discussion

The proposed architectures are tested and verified using the MRI images and video sequences shown in Figure 4.19. The proposed architectures are implemented on a Virtex5 5vlx50tff1136-3 Xilinx device using the Xilinx system generator tool [134]. The output data and memory addressing are validated according to the results obtained by specific Matlab codes. Matlab codes for 3-D DCT has been produced and used as a test bench for 3-D input and output data. For testing purposes the parameters IWL and FWL have been chosen to be (12, 9), (12, 6) and (12, 4) for total WL of 21, 18 and 16-bit, respectively.

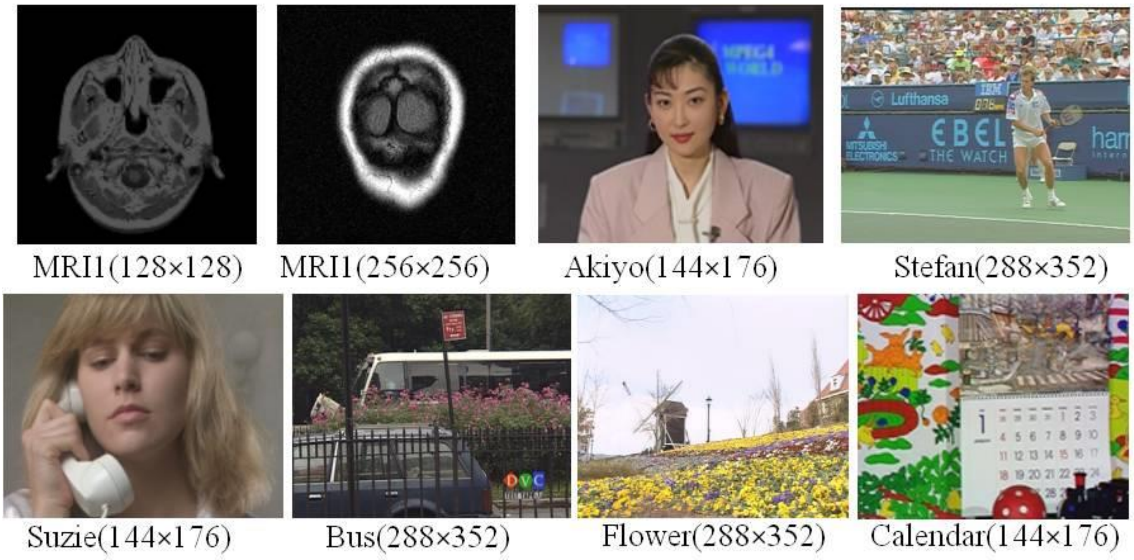


Figure 4.19: Test MRI images and video sequences (first frame from each sequence).

Further, the proposed architectures have been designed and implemented using Xilinx System generator tool, as illustrated in section 2.8. The initialisation Matlab codes for the proposed architectures are listed in the Appendix D.

4.5.1 Rate Distortion Performance

The PSNR and RMSE between the original and the reconstructed frames are computed using 21, 18 and 16-bit WL, as shown in Table 4.3. Further, the maximum absolute error between the 3-D DCT coefficients obtained by the proposed architectures and Matlab implementation are listed in Table 4.3 in columns 4 and 6 for RCF and 3-D DCT VR architectures, respectively. All figures in Table 4.3 represent the results that are obtained for eight frames from each data sequence. The $RMSE$ and $PSNR$ between the original and the reconstructed frames are computed using (2.12) and (2.3), respectively.

Moreover, the maximum absolute error between the proposed architectures and a Matlab implementation of the 3-D DCT is computed as follows:

$$MaxErr(n) = max\left(abs\big(DCT_M(i,j,n) - DCT_A(i,j,n)\big)\right)$$
$$and\ \overline{MaxErr} = \frac{1}{F}\sum_{n=1}^{F} MaxErr(n) \tag{4.25}$$

where $MaxErr(n)\ and\ \overline{MaxErr}$ are the maximum absolute error in each frame and the average of maximum absolute error over eight frames, respectively. Further, the variables DCT_M and DCT_A represent the 3-D DCT coefficients of each data cube computed using Matlab code and the proposed architectures, respectively.

From Table 4.3, it is obvious that at a wordlength of 21-bit, a $PSNR$ of ∞ has been achieved. The maximum average absolute errors among the eight input sequences are 3.11 and 0.87×10^{-2} for RCF and 3-D DCT VR architectures, respectively. Furthermore, a 51.2 and 52.9 dB $PSNR$ are achieved with 16-bit wordlength for both architectures, respectively. It can be explained that the error increasing in RCF over 3-D DCT VR architectures is a result of the high number of multipliers in each stage, which is accompanied by accumulated rounding and truncation operations.

Table 4.3: The PSNR and RMSE between original and reconstructed input sequence and the maximum absolute error in 3-D DCT coefficients for the first eight input frames.

Input sequence	RCF architecture			3-D DCT VR architecture		
	Reconstructed data		Coefficients	Reconstructed data		Coefficients
	PSNR (dB)	RMSE	$\overline{\text{MaxErr}}$ $\times 10^{-2}$	PSNR (dB)	RMSE	$\overline{\text{MaxErr}}$ $\times 10^{-2}$
			WL=21-bit			
MRI1	∞	0	1.18	∞	0	0.69
MRI2	∞	0	2.71	∞	0	0.87
Akiyo	∞	0	2.06	∞	0	0.70
Stefan	∞	0	2.53	∞	0	0.80
Suzie	∞	0	2.20	∞	0	0.71
Bus	∞	0	3.11	∞	0	0.83
Flower	∞	0	2.64	∞	0	0.87
Calendar	∞	0	2.33	∞	0	0.78
Average	∞	0	2.35	∞	0	0.78
			WL=18-bit			
MRI1	68.02	0.10	6.57	70.48	0.08	5.30
MRI2	65.18	0.14	7.22	67.81	0.10	5.69
Akiyo	66.77	0.12	6.81	69.38	0.09	5.30
Stefan	65.30	0.14	7.48	67.78	0.10	5.89
Suzie	65.28	0.14	7.16	67.83	0.10	5.68
Bus	65.21	0.14	7.54	67.76	0.10	6.10
Flower	65.33	0.14	7.72	67.78	0.10	6.06
Calendar	65.24	0.14	7.35	67.80	0.10	5.59
Average	65.79	0.13	7.23	68.33	0.10	5.70
			WL=16-bit			
MRI1	53.34	0.55	27.46	55.13	0.45	21.32
MRI2	50.45	0.77	29.75	52.39	0.61	23.52
Akiyo	52.93	0.58	26.64	54.15	0.50	21.87
Stefan	50.59	0.75	29.75	52.41	0.61	23.82
Suzie	50.66	0.75	27.74	52.47	0.61	21.87
Bus	50.51	0.76	29.72	52.39	0.61	23.57
Flower	50.75	0.74	29.31	52.46	0.61	23.49
Calendar	50.45	0.77	28.37	52.37	0.61	22.48
Average	51.21	0.71	28.59	52.97	0.58	22.74

4.5.2 Hardware Usage

Table 4.4, shows the hardware usage of the proposed architectures and the 3-D DCT architectures in [25]. This table shows that the proposed architectures consumes far fewer hardware resources than the 3-D DCT architectures in [25]. As such, 128 multipliers are required in [25] while only four multipliers are required in the 3-D DCT VR architecture for a cube size of 8×8×8-point. Further, the 3-D DCT VR architecture outperforms the RCF architecture in terms of slice registers, as detailed in Table 4.4. The number of slice registers used in the RCF architecture is increased due to the pipelining process to improve the operating frequency and to preserve data synchronisation through each 1-D DCT stage. However, the proposed 3-D DCT VR architecture consumes more memory resources than the RCF architecture, and this occurred because of the reordering and the data addressing process that is required by each stage of the proposed 3-D DCT VR architecture.

In addition, the average hardware utilisation rate for the RCF and 3-D DCT VR architectures is 15 and 13 % respectively from the available resources as shown in Table 4.5. Moreover, from this table, it is obvious that the number of DSP slices and block memory are lower than that required by the 3-D DCT VR architecture. However, the 3-D DCT VR has lower number or registers and LUTs.

Table 4.4: Comparison between hardware usages for different 3-D DCT architectures

Slice Logic Utilisation	Proposed architectures WL=21 bit		Architectures in [25]				
	RCF	3-D DCT VR		Original(1)	Pipelined ver.1(1)	Pipelined ver. 2(1)	Block
Slice Registers	6,934	4,972	Registers	57,379	57,367	105,476	13,347
Slice LUTs	7,051	2,779	Logic Elements(2)	117,847	184,396	306,188	19,355
Bonded IOBs	30	30	I/O Pins	2,059	2,059	2,058	525
Multipliers	39	4	9-bit Multipliers	576	576	576	128
Data cube size	8×8×8		8×8×8				
Target FPGA Device	XilinxVirtex5 5vlx50tff1136-3		Altera Cyclone III EP3C120F780C8				

(1) Estimated values were evaluated using Altera Quartus II [25].

(2) Each logic element in Altera Cyclone III comprises a four-input LUT [135].

Table 4.5: Comparison between percentage utilisation rates of the proposed architectures using 21, 18 and 16-bit wordlengths.

Slice Logic Utilisation	Available	Utilisation rates %					
		RCF			3-D DCT VR		
		21-bit	18-bit	16-bit	21-bit	18-bit	16-bit
Number of Slice Registers	28,800	24	20	18	**17**	**15**	**14**
Number of Slice LUTs	28,800	24	19	18	**9**	**8**	**8**
Number of occupied Slices	7,200	31	26	23	21	19	16
Number of bonded IOBs	480	6	5	5	6	5	5
Number of Block RAM/FIFO	60	**5**	**5**	**5**	15	15	15
Number of DSP48Es	48	**0**	**0**	**0**	12	12	12
Average utility rate %	--	15	13	12	13	12	12

4.5.3 Speed and Power Consumption

The maximum operating frequencies, computation times and power consumption of the proposed architectures using different clock periods and wordlengths are shown in Table 4.6.The maximum operating frequency of the RCF architecture is 237, 279 and 237 MHz for 21, 18 and 16 bit wordlengths, respectively. Whereas, the 3-D DCT VR architecture can operate up to 259, 238 and 305 MHz using the same wordlengths, respectively. The obtained results have revealed that for a data of 512×512×8-pixel the 3-D DCT can be computed for just 6.8 ms. Further, the dynamic power consumption of the proposed RCF architecture at 5 ns clock period are 261, 228 and 179 mW for 21, 18 and 16 bit wordlengths, respectively, whereas the proposed the 3-D DCT VR architecture can operate at 5 ns, consuming a dynamic power of 180, 153 and 129 mW for the same wordlengths, respectively.

It is clear from Table 4.6 that the 3-D DCT VR architecture consumes less power than the RCF architecture. The reduction in power consumption is achieved due to using a lower number of multipliers in 3-D DCT VR architecture (one multiplier per butterfly stage only) and using efficient data controllers. In contrast, the reasons for the high power consumption with the RCF architecture are additional multipliers and registers that are required to improve the speed and maintain the data synchronization at each 1-D DCT stage. The data controllers are designed to avoid using long delay elements and turn on the memory writing/reading operations only when required. In contrast, the reasons for the high power consumption that exhibit in the RCF architecture are that the relatively high number of multipliers and the additional registers required to improve the speed and maintain data synchronisation at each 1-D DCT stage.

Table 4.6: The power consumption and computation speed of the proposed architectures using different clock frequencies and 21, 18 and 16-bit wordlengths.

	Clock Frequency (MHz)	RCF Architecture			3-D DCT VR Architecture		
		21-bit	18-bit	16-bit	21-bit	18-bit	16-bit
Power consumption (mW)	200	261	228	179	180	153	129
	100	157	133	99	98	86	83
	66.67	115	100	72	71	64	62
	50	96	83	58	58	53	52
	40	84	73	50	51	47	46
	33.33	76	67	45	45	42	41
Operating frequency (MHz)		237	279	237	259	283	305
Computation time of 512×512×8-point (ms)		8.85	7.52	8.85	8.10	7.41	6.88

In addition, a comparison between the maximum operating frequency achieved by the proposed architectures with the architectures in [102] and [25] is shown in Table 4.7. As these architectures were implemented on different hardware devices, thus normalised operating frequencies are used for such comparison. Taking the maximum clock frequency of 5vlx50tff1136-3 Virtex5 Xilinx FPGA device as a reference, the frequency normalisation operation can be computed as follows [25]:

$$Normlised\_Freq1 = Freq1 \times \frac{Max.clock\ of\ 5vlx50t}{Max\_Clock\_Of\_Platform\_A} \qquad (4.26)$$

where, *Normalised_Freq1* is the normalised value of the architecture operating frequency *Freq1* which is implemented on hardware device type A. The variable *Max_Clock_Of_Device_A* represents the maximum clock frequency of the hardware device A. It is equal to 180 and 402.5 MHz for Altera EPF10K100EFC484 and Cyclone II EP2C70F896C6, respectively [25]. From Table 4.7, it is clear that the proposed architectures outperform those architectures in term of the normalised clock frequency.

Table 4.7: Comparison of the normalised operating frequencies for different 3-D DCT architectures.

	Proposed architectures WL=21-bit		Architecture in [102]	Architectures in [25]			
	RCF	3-D DCT VR		Original	Pipelined ver.1	Pipelined ver.2	Block
Frequency (MHz)	237	259	12.5	97.2	111.7	97.2	128.6
Normalised frequency	237	259	38.19	132.82	152.63	132.82	175.73
Data Cube	8×8×8		4×4×4	4×4×4			
Target hardware	Xilinx Virtex5 5vlx50tff1136-3		Altera EPF10K100EFC484	Altera Cyclone II EP2C70F896C6.			

4.6 Summary

This chapter has presented two new FPGA architectures for high speed 3-D DCT computation. The proposed architectures are based on 1-D DCT Radix-2 and 3-D DCT VR algorithms. The first architecture performs the 3-D DCT computation by applying 1-D DCT Radix-2 to the rows, columns and frames. Both architectures have been implemented, tested and evaluated using a Virtex5 Xilinx FPGA device. The obtained results have revealed that the RCF and 3-D DCT VR architectures can operate at up to 279 and 305 MHz respectively, using an 8×8×8-point data cube size. At such high speeds, the 3-D DCT computation times of 512×512×8-point are 7.5 and 6.8 ms, respectively. Moreover, the hardware usage of the 3-D DCT VR architecture is lower than that required by the RCF architecture, whereas the 3-D DCT VR architecture consumed more DSP48E slices and memory resources than the RCF architecture. The performance evaluation has revealed that the power consumption of the 3-D DCT VR and RCF architectures at 21-bit wordlength and 10 ns clock are 98 and 157 mW, respectively. Moreover, for different wordlengths, both architectures produce low error in the 3-D DCT output and good PSNR. The proposed architectures outperform similar 3-D DCT architectures in terms of the number of multipliers, I/O pins and registers. However, although the proposed 3-D VR architecture gained more advantages than that achieved by RCF, the main disadvantages of the 3-D VR architecture were the complicated memory addressing and high hardware usage. These drawbacks will be solved by the proposed architectures that will be presented in chapter 5.

Chapter 5: Area-Efficient 3-D DCT Architectures

5.1 Introduction

In this chapter, two new area-efficient architectures for 3-D DCT computation using the 3-D DCT VR algorithm are introduced. The data processing of the first architecture is accomplished sequentially with one output/stage each clock cycle, whereas two outputs/stages are computed in the butterfly and post addition stages in the second architecture. This has the merit of increasing the processing speed, albeit at the cost of extra hardware resources. The main characteristics of the proposed architectures are:

- In place computation; no block memory is used for data transposition in post-addition and butterfly computation stages which is essential in row-column-frame approaches. Thus, efficient architectures in terms of processing speed and power consumption are produced.
- Parameterisable architectures regarding wordlength, providing different output precision levels and processing speeds.
- Low hardware usage and high processing speed of up to 330 MHz using 14-bit output wordlength and 8×8×8-pixel input cube size can be achieved.

The proposed architectures have been tested on MRI images and video data sequences using different wordlengths. The results obtained show that the hardware usage is less than 10 and 16 % from the available resources of the Virtex5 5vlx50tff1136-3 FPGA device for the first and second architectures, respectively. Furthermore, an operating frequency of up to 330 MHz and a processing time of just 6.4 ms can be achieved for 512×512×8-pixel data using a transform length of 8×8×8.

The rest of this chapter is organised as follows. Section 5.2 presents an overview about the proposed architectures. Section 5.3 introduces a single path data flow architecture; Model1. While a dual path data flow architecture; Model2 is introduced in section 5.4. A sample of the results obtained is discussed in section 5.5 and the chapter summary is given in section 5.6.

5.2 The 3-D DCT VR Architectures

The proposed architectures consist of 3-D reordering, butterfly computation, post addition and 3-D bit reverse ordering stages, as shown in Figure 5.1. The input data is partitioned into cubes of $N \times N \times N$-voxel; N=8, or eight 8×8-pixel blocks and the dimension of the input sequence is $P \times Q \times F$-pixel. In the rest of this chapter, the term data cube refers to 8×8×8 or N^3-pixel, data block refers to 8×8 or N^2-pixel and the symbol B_i refers to an 8×8-pixel data block, $i=0,1,2,\ldots 7$ (B_0, B_1, ... B_7). As shown in Figure 5.2.

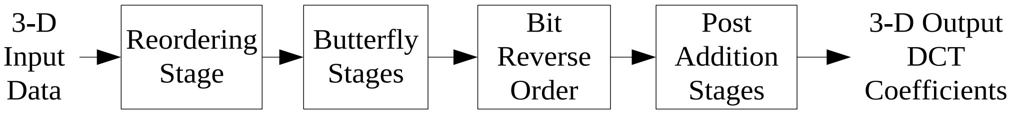


Figure 5.1: Block diagram of the 3-D DCT VR algorithm.

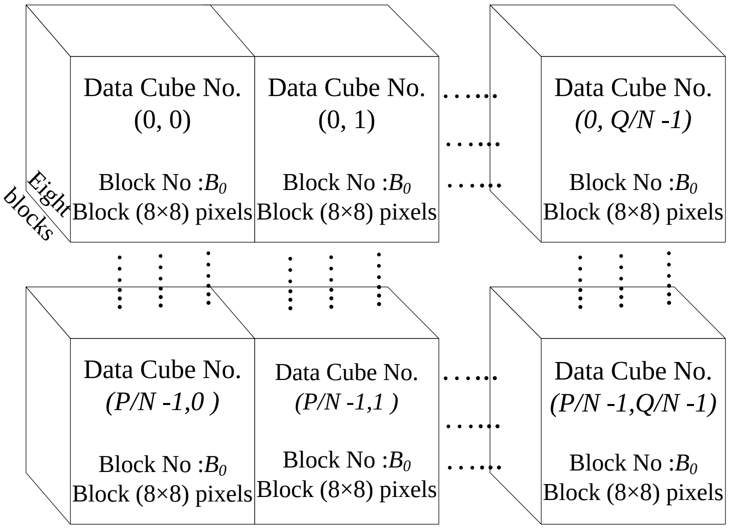


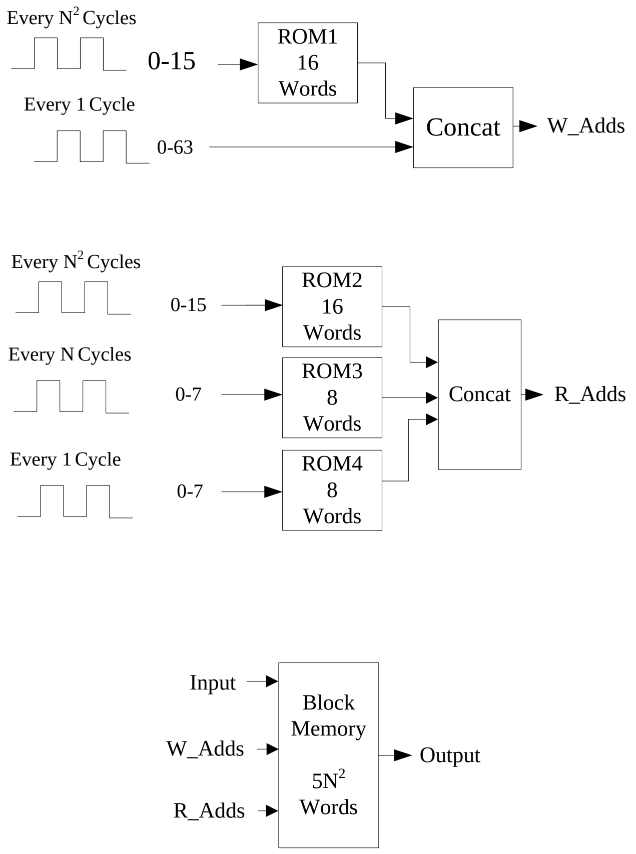
Figure 5.2: An 8×8×8-pixel data cubes.

Two models are proposed for 3-D DCT VR computation: Model1; single path data flow architecture, and Model2; dual path data flow architecture. The differences between the two models are in butterfly computation, post addition and 3-D bit reverse order stages. However, the same 3-D reordering circuit is used in both models. Furthermore, the wordlength of the proposed architectures can be adjusted using two input parameters; the IWL and FWL.

5.2.1 Reordering Stage

The proposed architectures partition the input sequence into cubes of 8×8×8-pixel or eight 8×8-pixel data blocks. The input data to the proposed architectures is reordered according to (4.8). The reordering process is performed by shuffling each element within each column, shuffling between columns and shuffling between blocks. As an example, if the input sequence is; 0, 1, 2, 3, 4, 5, 6 and 7, then the reordered sequence will be; 0, 2, 4, 6, 7, 5, 3 and 1. The block diagram of reordering stage is shown in Figure 5.3.

From Figure 5.3, the reordering circuit composes of a $5N^2$-word block RAM and read/write controllers. During the first four N^2 clock cycles, the first four data blocks are allocated in the first $4N^2$ memory locations using its natural order from 0 to 255. At the beginning of the fifth block (i.e. the element number 256) the memory reading process starts to perform the reordering process. The proposed writing/reading sequence is shown in Table 5.1. From this table, the delay between memory writing and reading operations is $4N^2$ clock cycles. The writing addresses are generated using 16-word ROM (ROM1) which direct each block into an empty memory location to avoid interference between writing and reading operations. The ROM1 content is; 0, 1, 2, 3, 4, 0, 2, 4, 2, 4, 0, 3, 1, 2, 0 and 1; where 0 refers to the memory locations from 0 to 63, and 1 refers to 64 to 127 locations and so on. Further, the memory reading addresses are generated using 16-words ROM (ROM2) to select the required data block: the reading phase ROM2 content is shown in the sixth column of Table 5.1. Moreover, two eight word ROMs (ROM3 and ROM4) are used to generate the reordering address within each 8×8-pixel block for row and column reordering. The content of each ROM is; 0, 2, 4, 6, 7, 5, 3 and 1. The output of this stage represents a reordered version of the input data cube which is fed to the subsequent stage; butterfly computation stage.



a. Reordering Circuit

Input Blocks/sequence from left to right

0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0
0	0	0	0		5	5	5	5		2	2	2		6		0
	1	1	1	1	1	1	1	1	1	1		4	4		7	
		2	2	2		6		0	0	0	0		5	5	5	5
			3	3	3	3	3	3	3		3	3	3	3	3	3
				4	4		7		1	1	1	1	1	1	1	1
Delay				0	2	4	6	7	5	3	1	0	2	4	6	7

Output Blocks

Each square represents (8×8) coefficients or one block

b. Memory content

Figure 5.3: Reordering circuit and memory content during reordering operation.

Table 5.1: The memory write/read block locations for reordering stage (block shuffling)

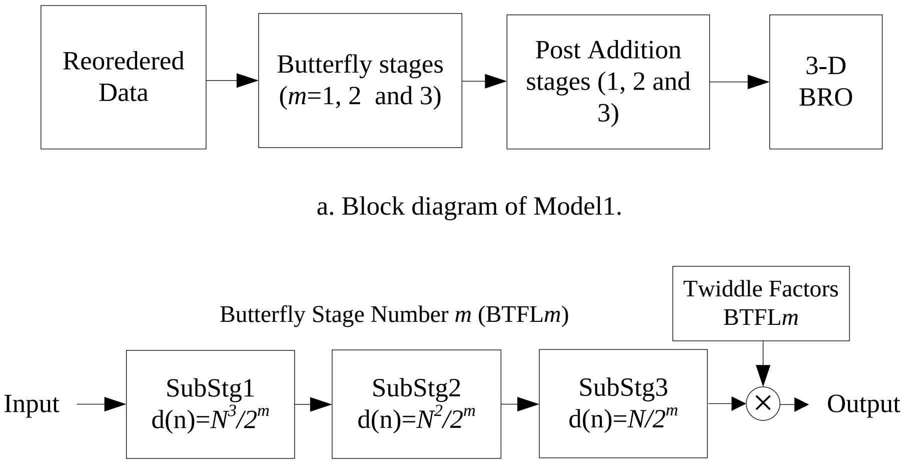
Input Data			Output Data		
Cube number	Block number	Memory addresses	Cube number	Block number	Memory addresses
0	B_0	**0**	--	--	--
0	B_1	**1**	--	--	--
0	B_2	**2**	--	--	--
0	B_3	**3**	--	--	--
0	B_4	**4**	0	B_0	**0**
0	B_5	**0**	0	B_2	**2**
0	B_6	**2**	0	B_4	**4**
0	B_7	**4**	0	B_6	**2**
1	B_0	**2**	0	B_7	**4**
1	B_1	**4**	0	B_5	**0**
1	B_2	**0**	0	B_3	**3**
1	B_3	**3**	0	B_1	**1**
1	B_4	**1**	1	B_0	**2**
1	B_5	**2**	1	B_2	**0**
1	B_6	**0**	1	B_4	**1**
1	B_7	**1**	1	B_6	**0**
2	B_0	**0**	1	B_7	**1**
2	B_1	**1**	1	B_5	**2**
2	B_2	**2**	1	B_3	**3**
2	B_3	**3**	1	B_1	**4**

5.3 Single Path Data Flow 3-D DCT Architecture; Model1

The proposed architecture (Model1) is composed of a 3-D reordering stage, three butterfly computation stages, three post addition stages and 3-D Bit Reverse Order (3-D BRO) stages. The reordering unit has been introduced in section 5.2.1, while the other stages are introduced in 5.3.1 to 5.3.3.

5.3.1 Butterfly Stages

The butterfly stage consists of three main stages, each with three sub-stages, as shown in Figure 5.4. In Figure 5.4-b, the letter *m* refers to the butterfly stage number; BTFL$_{1}$, BTFL$_{2}$ and BTFL$_{3}$. Every butterfly stage consists of three single units, a twiddle factor generator and a multiplier. Each sub-stage is considered as a single butterfly unit (single sub-stage), as shown in Figure 5.5.



a. Block diagram of Model1.

b. Single Butterfly stage.

Figure 5.4: Block diagrams of a. Model1 and b. Butterfly architectures.

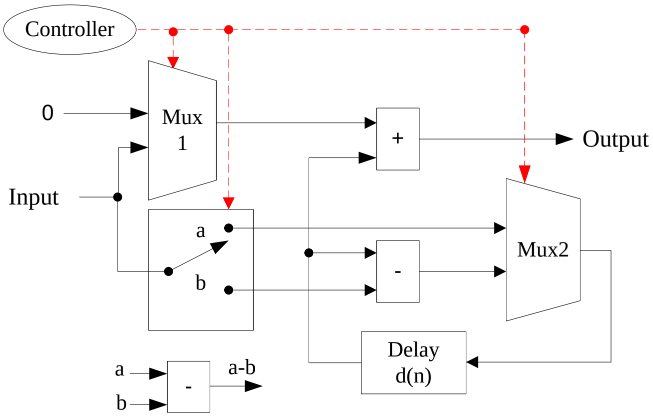


Figure 5.5: Single butterfly unit of Model1.

The first butterfly unit; BTFL$_1$-SubStg1 is used to perform the addition and subtraction between the two halves of each input data cube; 0 to $\frac{N^3}{2}-1$ and $\frac{N^3}{2}$ to N^3-1. During the first $N^3/2$ samples of each data cube, the controller is in its low state; 0. This enforces the output of the first multiplexer; Mux1 to 0, and the switch is connected to the upper side; point-a, and the output of the second multiplexer; Mux2 represents the first half of the input data cube. This arrangement allows the first half of the input cube to be allocated in $N^3/2$ registers; *d(n)*=$N^3/2$. However, in the succeeding $N^3/2$ clock cycles, the controller state is turned into high level; 1 to allow the second half of the input data cube to go through to the adders. The output of the Mux2 during this period is the result of subtraction between the first and second halves of the input data cube and the switch connected to the lower side; point-b. The output data represents the additions between the two halves of the input data cube. Moreover, during the first half of the second data cube the controller will return to its low state; 0 to allow the subtraction results of the first data cube to go through the output port. Consequently, the output data from BTFL1-SubStg1 is kept in its original order; 0 to N^3*-1* and the initial output latency is $N^3/2$.

In the next sub-stage; $BTFL_1$-SubStg2, the controller period becomes $N^2/2$ clock cycles to allow addition and subtraction between the two halves of each block; N^2 samples. The same computation procedure in the SubStg1is used to perform the addition and subtraction process in SubStg2. Finally, the third sub-stage of the first butterfly; $BTFL_1$-SubStg3 acts within each single column; eight samples to perform addition and subtraction between the two halves of each column.

The circuit operation is directed using a specific controller which is composed of three single-bit slices from the main counter. The time period of the three controlling signals are: $N^3/2$, $N^2/2$ and $N/2$ clock cycles.

The output of $BTFL_1$ stage is multiplied by an appropriate twiddle factor using single multiplier and twiddle factor generator circuit. The $BTFL_1$ twiddle factor generator; TF-$BTFL_1$ composes of three eight word ROMs, two multipliers and a specific addressing signal, as shown in Figure 5.6. The contents of each ROM are; 1, 1, 1, 1, 1.96157, 1.11114, -0.39018 and -1.66294 and they are divided by 2 to prevent output overflow. The addresses of ROM1, ROM2 and ROM3 are generated every one, *N and* N^2 clock cycles, respectively, as shown in Figure 5.6.

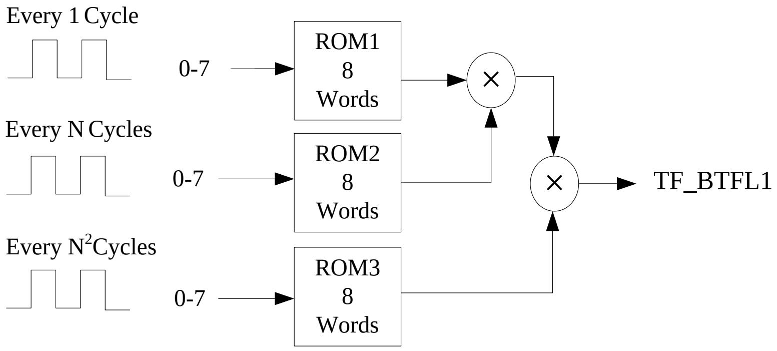


Figure 5.6: Twiddle factor generator circuit of the first butterfly stage.

The second butterfly stage; BTFL2 composes of three sub-stages, similar to BTFL1 with a different registers, data controlling signals and twiddle factors. The required registers for the second butterfly stage are $N^3/4$, $N^2/4$ and $N/4$ for the three sub-stages, respectively, as shown in Figure 5.4. Moreover, the three controlling signals are generated using single-bit slices from the main counter considering that the required time periods of the three signals are; $N^3/4$, $N^2/4$ and $N/4$, respectively.

Further, the twiddle factors of BTFL2 are generated similar to BTFL1 using four word ROMs, two multipliers and data addressing units, as shown in Figure 5.7. The content of each 4-word ROMs is; 1, 1, 1.847759 and -0.76537, divided by 2 to reduce the number of bits required for the IWL part of the output samples. Again, in this stage the output is normalised by 8 and as a result the final output becomes normalised by 64; 8 for BTFL1 and 8 for BTFL2.

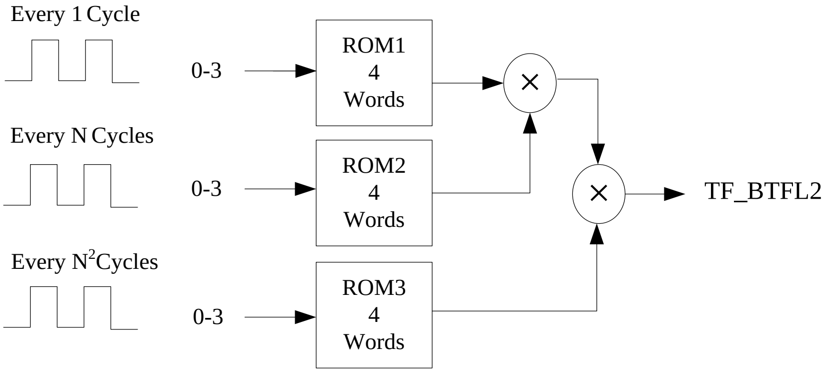


Figure 5.7: Twiddle factor generator circuit of BTFL2.

The last butterfly stage; BTFL3 is similar to the previous two stages with different registers, data controller, twiddle factors and the third sub-stage. The first two sub-stages in BTFL3 are similar to those used in BTFL1 and BTFL2 but with $N^3/8$ and $N^2/8$ registers; *d(n)* for the first and second sub-stage, respectively. On the other hand, the third sub-stage is constructed in a different manner using one multiplexer, two adders, two down sampler elements and one register, as shown in Figure 5.8. Further, the data controlling signals are generated using three single-bit slices with a time period equal to $N^3/8$, $N^2/8$ and $N/8$ cycles.

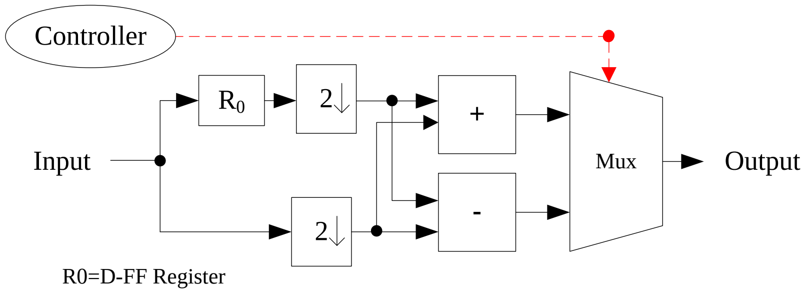


Figure 5.8: Last sub-stage of BTFL3.

The output of BTFL3 is multiplied by an appropriate twiddle factors using a single multiplier. Twiddle factors are generated using single four words ROM and three single-bit slices and two adders, as illustrated in Figure 5.9. The ROM content is; 1, 1.414214, 2 and 2.828427 normalised by 2, and the ROM addressing signal is generated using three single-bit slices. The generated twiddle factors are multiplied accordingly with each BTFL3 output using a single multiplier. The output from the third butterfly represents a normalised version of the regular butterfly output. The normalisation factor is 128; 8 for BTFL1, 8 for BTFL2 and 2 for BTFL3. The output of the butterfly stage BTFL3 is fed to the subsequent post-addition stages.

Briefly, a complete in-place computation is carried out for the butterfly stages without any reordering within or between the butterfly stages. Such computation procedure reduces the complexity of the proposed architecture.

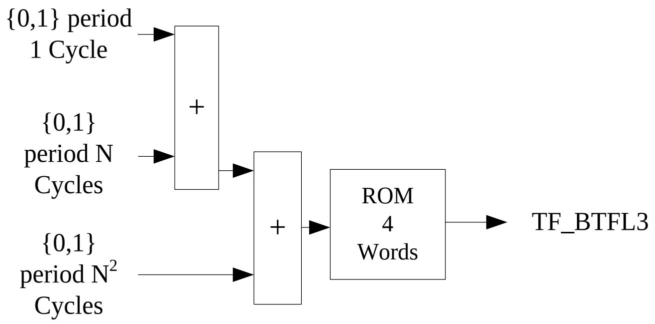
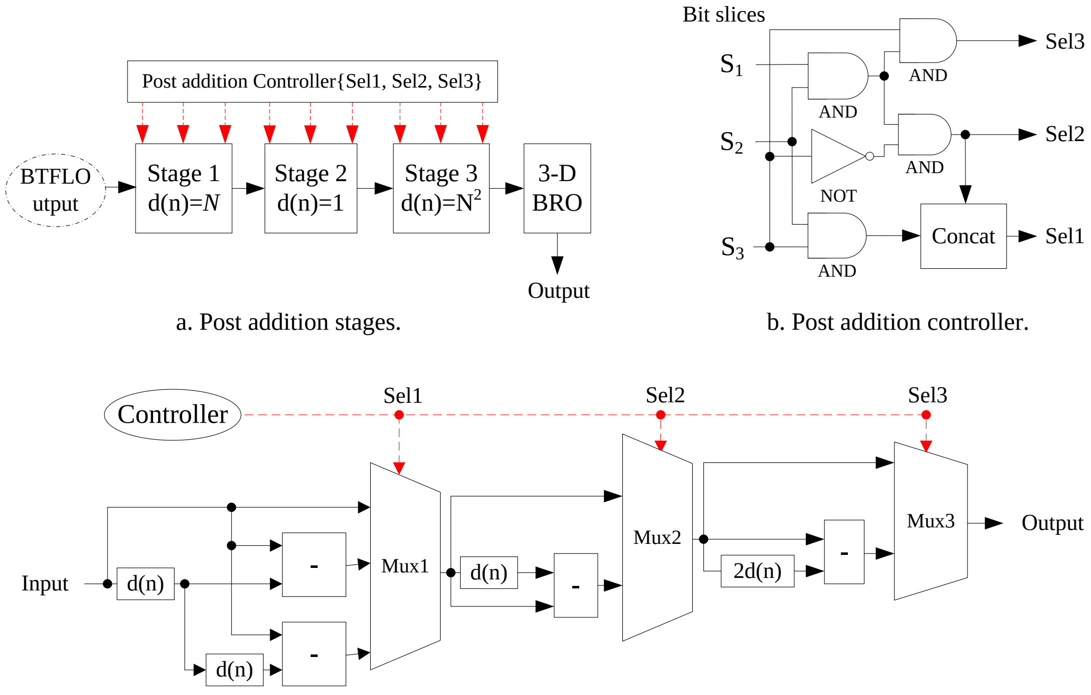


Figure 5.9: Twiddle factor generator circuit of BTFL3.

5.3.2 Post Addition Stages

The third part of the Model1 is the post addition stages, which perform the computation of the term outside the curly brackets of (4.13) to (4.19). The post addition operation composes of three stages; each stage runs an addition process for selected elements over each data dimension; Row, Column and Frame. The block diagram of the three post addition stages and their data controller are shown in Figure 5.10. Where d(n) is the register length, Sel1, Sel2 and Sel3 are the controller signals. From Figure 5.10-c, each post addition stage consists of three multiplexers, four adders, registers and controller units.



a. Post addition stages.

b. Post addition controller.

c. Single post addition stage of Model1

Figure 5.10: Post addition stages and their controller.

In the first post addition stage; Stage 1, each register is of *N* length; *d(n)*=8. Further, the data controller of the post addition stages is shown in Figure 5.10-b.

The output signals of the three controllers can be computed as follows:

$$Sel_3 = S_1 \bullet S_2 \ \bullet \ S_3 \tag{5.1}$$

$$Sel_2 = S_1 \bullet S_2 \ \bullet \ \overline{S_3} \tag{5.2}$$

$$Sel_1 = Concat\ \{Sel_2, (S_2 \bullet S_3)\ \} \tag{5.3}$$

where • represents logical AND, S_1, S_2 and S_3 represent different single-bit slices from the main counter and Sel_1, Sel_2 and Sel_3 are the resultant three controlling signals.

The selected bits for the first controller circuit (first post addition stage controller) are: S_1=C_5, S_2=C_4, S_3=C_3, where C is the binary representation of a 9-bit counter (LSB marked with 0 and MSB marked with 8).

The second post addition stage; Stage 2 is constructed similar to the Stage1. The main differences between them are register length and data controller. The register length for this stage is 1; *d(n)*=1, and the second controller signals are computed using (5.1) to (5.3) and the selected bits for the controller are $S_1=C_2$, $S_2=C_1$ and $S_3=C_0$.

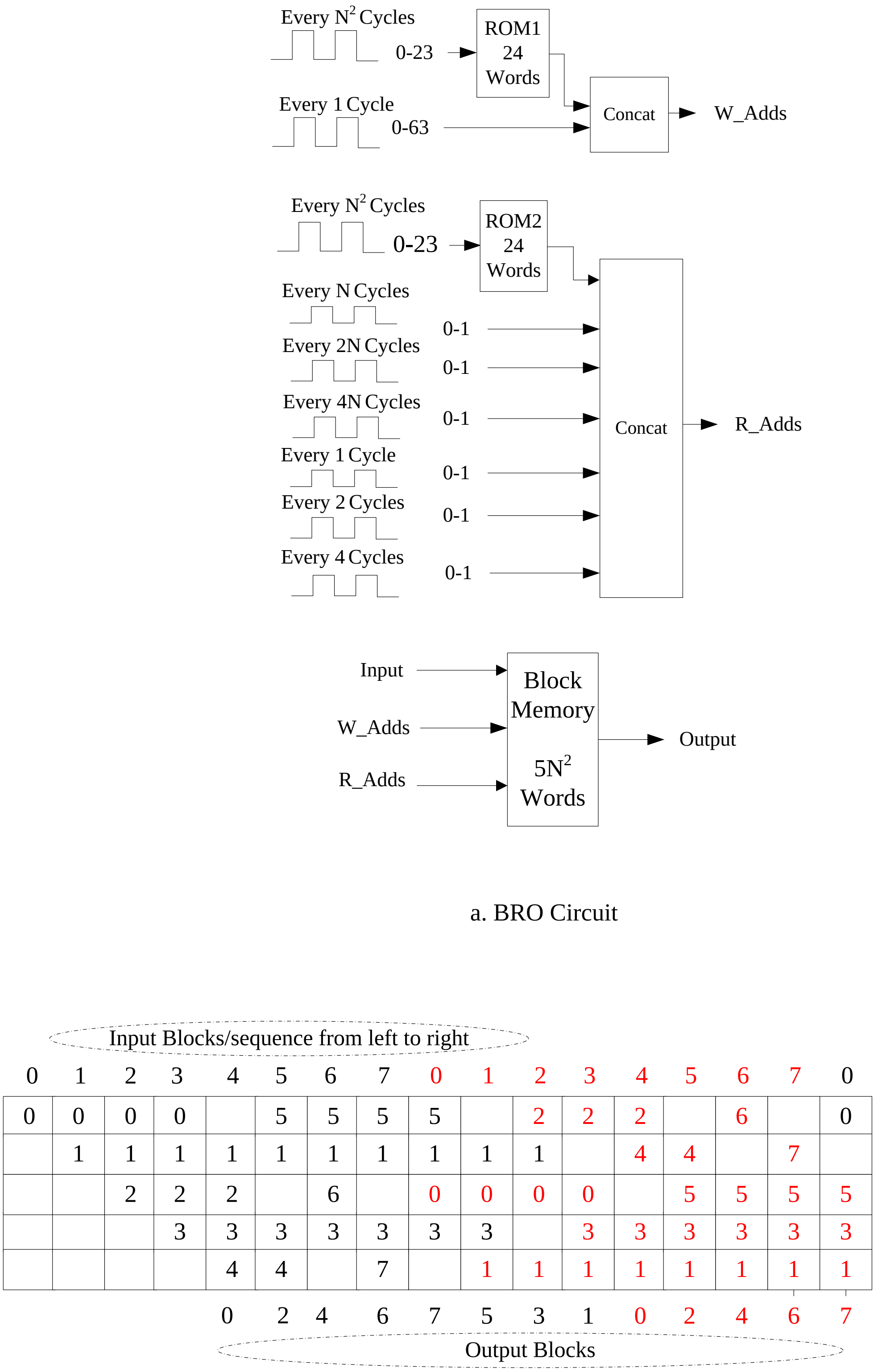
Further, the last post addition stage; ; Stage 2 is similar to the first and second stages but using N^2 registers for each sub-stage; *d(n)*=64 and the selected bits used to generate the controller signals are; $S_1=C_8$, $S_2=C_7$ and $S_3=C_6$. The output from the third post addition stage is fed to the 3-D bit reverse order stage to produce the same order of the input data. Further, the features of the post addition stages are: in place computation, and low complexity derived by simple controller units.

5.3.3 3-D Bit Reverse Order Stage (3-D BRO Stage)

The last stage of the proposed architecture is the 3-D BRO stage. This stage is postponed after the post addition stages to act as a buffer for the subsequent system as an example; DCT is usually integrated with a Quantiser in conventional data compression algorithms. The 3-D BRO stage composes of $5N^2$-word block memory and writing/reading addresses controller, as shown in Figure 5.11. The writing addresses are generated using 24-word ROM (ROM1) and 0-63 counter. ROM1 is used to specify an empty memory location of each data block. The ROM1 content is shown in the first 24 elements of the third column of Table 5.2. The input data blocks are written into its locations according to the order specified by ROM1 content in the same manner of reordering stages in sub-section 5.2.1.

During the memory reading phase, specific addresses are generated to perform the bit reversing operation. The bit reversing operation shuffles the regular input sequence into different arrangements; as an example for eight points; 0, 1, 2, 3, 4, 5, 6 and 7, the 1-D BRO sequence is; 0, 4, 2, 6, 1, 5, 3 and 7. The 3-D BRO operation is performed using 24-word ROM (ROM2) for frame direction shuffling and six single-bit slices from the main counter. ROM2 content is shown in the sixth column of Table 5.2, which is used for shuffling between data blocks within each data cube. The whole 3-D BRO addresses (memory reading addresses) are generated by concatenating ROM2 content with the six single-bit slices, as shown in Figure 5.11-a. The block memory content during each N^2 clock cycles is shown in Figure 5.11-b.

Finally, the output from this stage represents the 3-D DCT coefficients of the input data.



a. BRO Circuit

Input Blocks/sequence from left to right

0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0
0	0	0	0		5	5	5	5		2	2	2		6		0
	1	1	1	1	1	1	1	1	1	1		4	4		7	
		2	2	2		6		0	0	0	0		5	5	5	5
			3	3	3	3	3	3	3		3	3	3	3	3	3
				4	4		7		1	1	1	1	1	1	1	1
				0	2	4	6	7	5	3	1	0	2	4	6	7

Output Blocks

b. Memory content

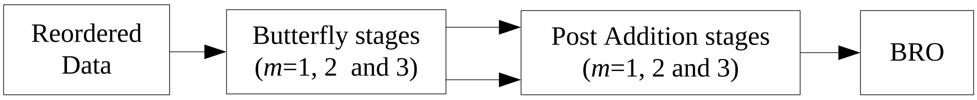
Figure 5.11: The 3-D BRO stage.

Table 5.2: The memory write/read block locations for the 3-D BRO stage (shuffling between data blocks)

Input Data			Output Data		
Cube number	Block number	Memory addresses	Cube number	Block number	Memory addresses
0	B_0	**0**	--	--	--
0	B_1	**1**	--	--	--
0	B_2	**2**	--	--	--
0	B_3	**3**	--	--	--
0	B_4	**4**	0	B_0	**0**
0	B_5	**0**	0	B_4	**4**
0	B_6	**4**	0	B_2	**2**
0	B_7	**2**	0	B_6	**4**
1	B_0	**4**	0	B_1	**1**
1	B_1	**1**	0	B_5	**0**
1	B_2	**0**	0	B_3	**3**
1	B_3	**3**	0	B_7	**2**
1	B_4	**2**	1	B_0	**4**
1	B_5	**4**	1	B_4	**2**
1	B_6	**2**	1	B_2	**0**
1	B_7	**0**	1	B_6	**2**
2	B_0	**2**	1	B_1	**1**
2	B_1	**1**	1	B_5	**4**
2	B_2	**4**	1	B_3	**3**
2	B_3	**3**	1	B_7	**0**
2	B_4	**0**	2	B_0	**2**
2	B_5	**2**	2	B_4	**0**
2	B_6	**0**	2	B_2	**4**
2	B_7	**4**	2	B_6	**0**
3	B_0	0	2	B_1	**1**
3	B_1	1	2	B_5	**2**
3	B_2	2	2	B_3	**3**
3	B_3	3	2	B_7	**4**

5.4 Dual Path Data Flow 3-D DCT Architecture; Model2

Model2 is dual path architecture for 3-D DCT VR computation. Model2 is designed to produce high speed 3-D DCT architecture with in-place butterfly computation procedure. It composes of a 3-D reordering stage, three butterfly stages, three post addition stages and a 3-D BRO stage, as shown in Figure 5.12. The 3-D reordering stage is similar to Model1, whereas different architectures of butterfly, post addition and BRO stages are introduced in this model. Each single butterfly and post addition stage computes two coefficients per clock cycle; the first output represents the first half of the input data (addition process) whereas the second output represents the second half of the input data (subtraction process).

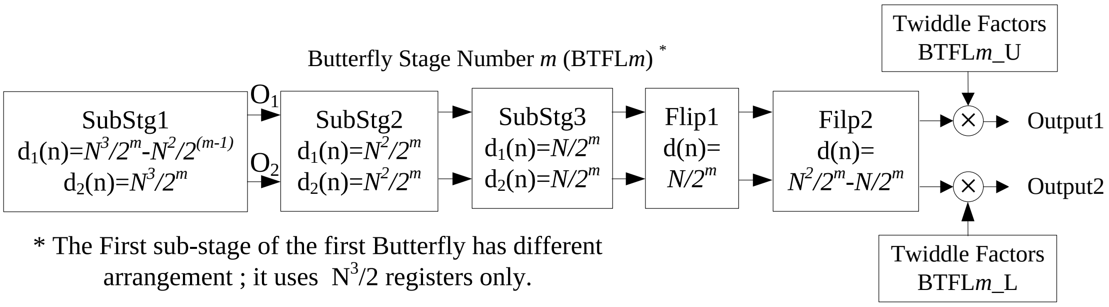


Reordered Data
Butterfly stages (m=1, 2 and 3)
Post Addition stages (m=1, 2 and 3)
BRO

Figure 5.12: a. Block diagram of the proposed dual path data flow 3-D DCT architecture; Model2

5.4.1 *Butterfly Stages*

The Model2 butterfly stages are shown in Figure 5.13. Two flipping stages are added to keep the data sequence within each block in its regular order; 0 to N^2*-1*.

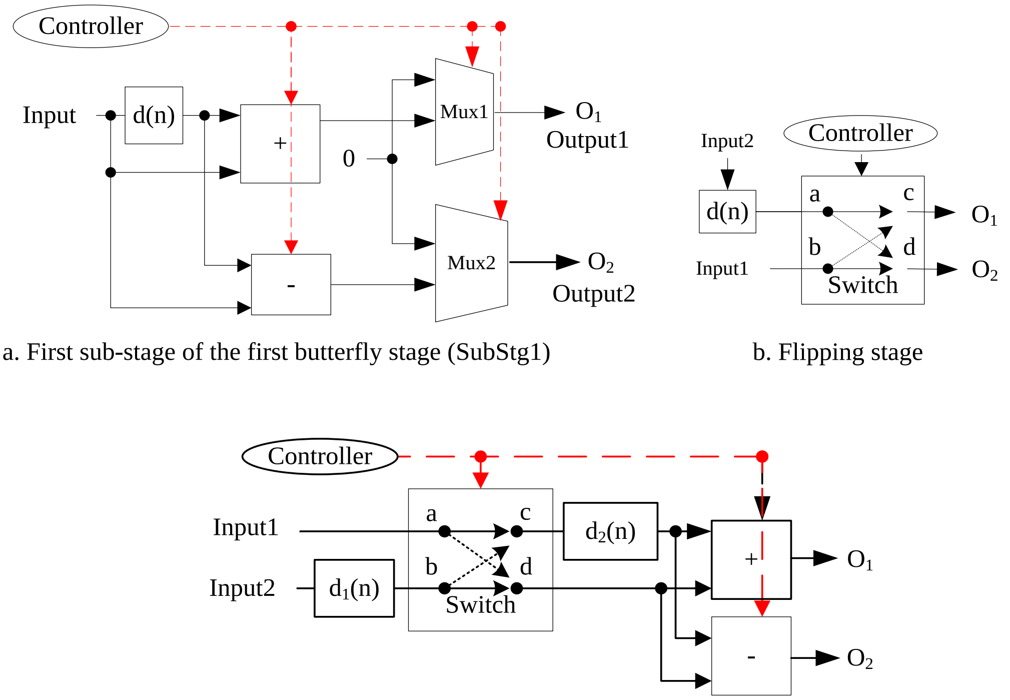


Butterfly Stage Number m (BTFLm) *
Twiddle Factors BTFLm_U
SubStg1 $d_1(n)=N^3/2^m-N^2/2^{(m-1)}$ $d_2(n)=N^3/2^m$
O_1
O_2
SubStg2 $d_1(n)=N^2/2^m$ $d_2(n)=N^2/2^m$
SubStg3 $d_1(n)=N/2^m$ $d_2(n)=N/2^m$
Flip1 $d(n)=N/2^m$
Filp2 $d(n)=N^2/2^m-N/2^m$
Output1
Output2
* The First sub-stage of the first Butterfly has different arrangement ; it uses $N^3/2$ registers only.
Twiddle Factors BTFLm_L

Figure 5.13: The architecture of single butterfly stage of Model2

The input data is firstly 3-D reordered and fed to the first sub-stage of the first butterfly stage; SubStg1 of BTFL1, $m=1$. The first sub-stage is constructed from two multiplexers, two adders and $N^3/2$ registers; $d(n)= N^3/2$, as shown in Figure 5.14-a. During the first $N^3/2$ clock cycles, the outputs of the two multiplexers (Mux1 and Mux2) are equal to 0 and the first $N^3/2$ pixels of the input data cube are stored in a specific register; $d(n)$; the register length is $d(n)= N^3/2$. However, during the second

$N^3/2$ clock cycles the output of Mux1 and Mux2 become the results of addition and subtraction operations, respectively. The addition and subtraction process is performed between the content of the register *d(n)* and the second half of the input data cube during the second $N^3/2$ clock cycles. The process of addition and subtraction in the first sub-stage produces $N^3/2$ clock cycles delay between successive data cubes, this delay will be eliminated in the last stage of Model2. The adder and subtracter components are turned off during the first half of each input cube, and on during the second half using the same controller signal of this stage. The first sub-stage controller signal is a bi-level signal, each level with a period of $N^3/2$ clock cycles, while $N^2/2$ and *N/2* clock period controller signals are used for the second and third sub-stages of BTFL1 stage.



a. First sub-stage of the first butterfly stage (SubStg1)

b. Flipping stage

c. Single butterfly sub-stage architecture

Figure 5.14: Butterfly sub-stages and flipping stage in Model2.

The data block order of the first sub-stage outputs are: first output order; O_1 is; B_0, B_1, B_2 *and* B_3 and the second output; O_2 is; *B4*, B_5, B_6 *and* B_7. The element order within each block is kept in its natural order from 0 to N^2*-1*.

The second sub-stage in $BTFL_1$ is constructed from a switch, two adders and two $N^2/2$-registers; $d_1(n)$, $d_2(n)=N^2/2$, as shown in Figure 5.14-c. During the first $N^2/2$, the switch connects the positions; a to c and b to d, and flips them during the successive $N^2/2$ clock cycles to connect the positions; a to d and b to c. The output O_1 represents the first half of each block, whereas the second half is handled by O_2. The block order of the two outputs is:

O_1 order; $[B_0^1, B_4^1, B_1^1, B_5^1, B_2^1, B_6^1, B_3^1, B_7^1]$

O_2 order; $[B_0^2, B_4^2, B_1^2, B_5^2, B_2^2, B_6^2, B_3^2, B_7^2]$ (5.4)

where B_{0-7}^1 represents the first half of each data block and B_{0-7}^2 represents the second half.

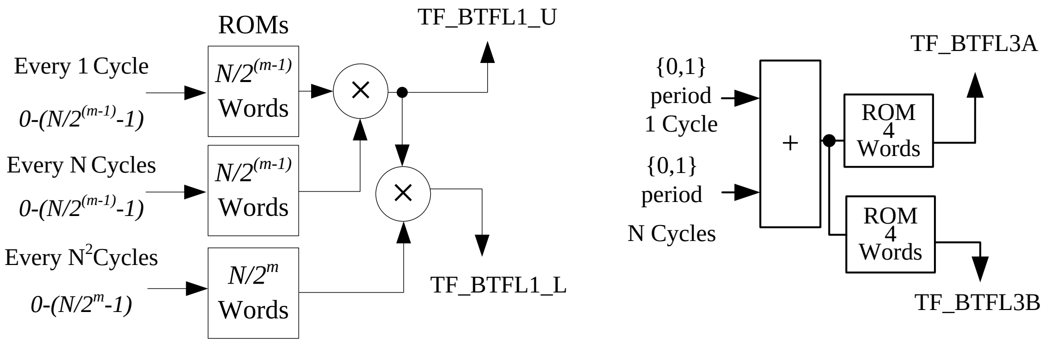
The circuit diagram of the third sub-stage in $BTFL_1$ is similar to the second sub-stage; the only difference is the length of the two registers, which is equal to $N/2$; $d_1(n)$, $d_2(n)$ $=N/2$. In this sub-stage, the order of the blocks at each output; O_1 and O_2 is (B_0, B_4, B_1, B_5, B_2, B_6, B_3 *and* B_7). However, the data order inside such blocks of O_1 and O_2 are:

O_1: 0-3, 32-35, 8-11, 40-43, 16-19, 48-51, 24-27 and 56-59

O_2: 4-7, 36-39, 12-15, 44-47, 20-23, 52-55, 28-31 and 60-63 (5.5)

This irregular element order within each data block is corrected using two flipping sub-stages. Such two sub-stages are constructed using a switch and registers, as shown in Figure 5.14-b. The first flipping circuit utilises a register length of $N/2$; $d(n)=4$ and the second flipping stage uses $N^2/2-N/2$ register; $d(n)=28$. The controller signals of the first and second flipping switches are the same as the controller signals of the third and second sub-stages of BTFL1, respectively.

The last part of the first butterfly stage is the twiddle factor multiplication, which is accomplished using one multiplier per output and a single twiddle factor generator circuit. The twiddle factor generator is shown in Figure 5.15-a. This circuit composes of two 8-word ROM; its content is the same as that in Model1, one 4-word ROM with a content of: 1.96157, 1.11114, -0.39018 and -1.66294 and two multipliers. The first ROM is derived by a 3-bit slice with a single clock cycle period and the second ROM is derived by a 3-bit slice with an N clock cycle period. The addresses of the third ROM are generated using a 2-bit slice with a clock period of N^2.



a. First (*m=1*)and second (*m=2*) stages twiddle factors generator

b. Third stage twiddle factors generator

Figure 5.15: Twiddle factors generators for butterfly stages.

In the second butterfly stage (BTFL2; *m=2*), all sub-stages are constructed using the circuit in Figure 5.14-c. The number of registers for each sub-stage are; (96,128), (16, 16) and (2, 2), where the first value of each pair represents $d_1(n)$ and the second value refer to $d_2(n)$). Three bi-levels controller signals are used to perform the butterfly computation; the period of the first signal is $N^3/4$ cycles, while the clock period of the second and the third signals is $N^2/4$ and *N/4*, respectively. Additionally, two flipping units are added with *d(n)*=2 and 14 for the first and second flipping units, respectively. Bearing in mind that the data within each block is kept in its natural order, the block order of the first output becomes; *B0, B1, B4 and B5*, and the second output order is; *B2, B3, B6 and B7*. The outputs from BTFL2 are multiplied by specific twiddle factors, accordingly. The twiddle factors of BTFL2 are generated using the circuit of Figure 5.15-a with *m=2*. The content of the upper and the middle ROM is; 1, 1, 1.847759 and -0.76537, while the content of the lower ROM is; 1.847759 and -0.76537.

The last butterfly stage; BTFL3 is similar to the previous stage but it uses different register lengths and twiddle factor values. The length of the registers (d_1, d_2) for each sub-stage are; (48, 64), (8, 8) and (1, 1) and the length of the registers used in the first and second flipping units are 1 and 7, respectively. The data block order of the first output O_1 is; *B0, B2, B4 and B6*, and the second output O_2 order is; *B1, B3, B5 and B7*, whereas the order of the elements within each block is corrected using the two flipping switches. Further, the twiddle factors of BTFL3 are computed using the circuit shown in Figure 5.15-b by using two ROMs. The content of both ROMs are; 1, 1.4142 and 2, and 1.4142, 2 and 2.8284, respectively.

5.4.2 Post Addition Stages

Post addition operations compose of three individual stages, as shown in Figure 5.16. Each post addition stage processes two elements per clock cycle. The first two stages are constructed from two parallel stages similar to the single post addition of Model1 stages, Figure 5.10-c; with the same data controller (Figure 5.10-b). On the other hand, the third stage is constructed using the circuit shown in Figure 5.17. The third post addition stage consists of four multiplexers, five adders and the registers length is N^2 ;$d(n)$=64.The controller of the last post-addition stage is composed of one AND gate and two single-bit slice elements, bit number 6 and 7 from the main 9-bit counter. The last adder is used to merge the two outputs into a single output; the order of the merged output is; B_0, B_2, B_4, B_6, B_1, B_3, B_5 and B_7.

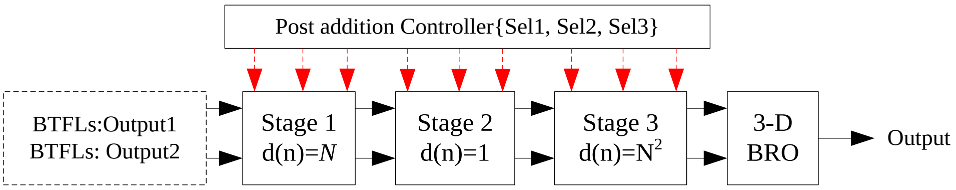


Figure 5.16: Post addition stages.

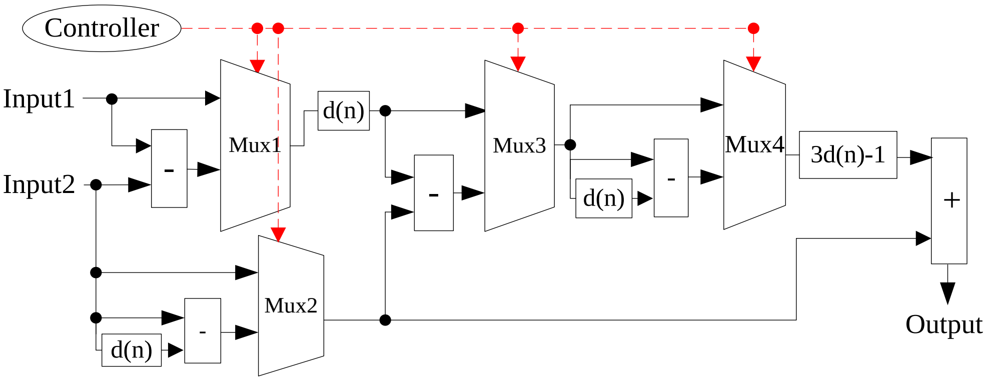


Figure 5.17: Third post addition stage (Stage3).

5.4.3 3-D Bit Reverse Order Stage (3-D BRO Stage)

The 3-D BRO stage of Model2 is similar to that of Model1, as shown in Figure 5.11. However, different block RAM and ROM1 and ROM2 sizes are used. The block memory size is $3N^2$–word, and ROM1 and ROM2 are of 8-word depth. The content of ROM1 is; 0, 1, 2, 0, 2, 1, 0 and 2, whereas; 0, 2, 1, 0, 2, 0, 1 and 2, is the content of ROM2.

5.5 Performance Evaluation

The proposed architectures have been tested and implemented on a Xilinx Virtex5 5vlx50tff1136-3 FPGA device, using different video sequences and different wordlengths. Further, the proposed architectures have been designed and implemented using Xilinx System generator tool, as illustrated in section 2.8. The initialisation Matlab codes for the proposed architectures are listed in the Appendix E.

5.5.1 Rate Distortion Performance

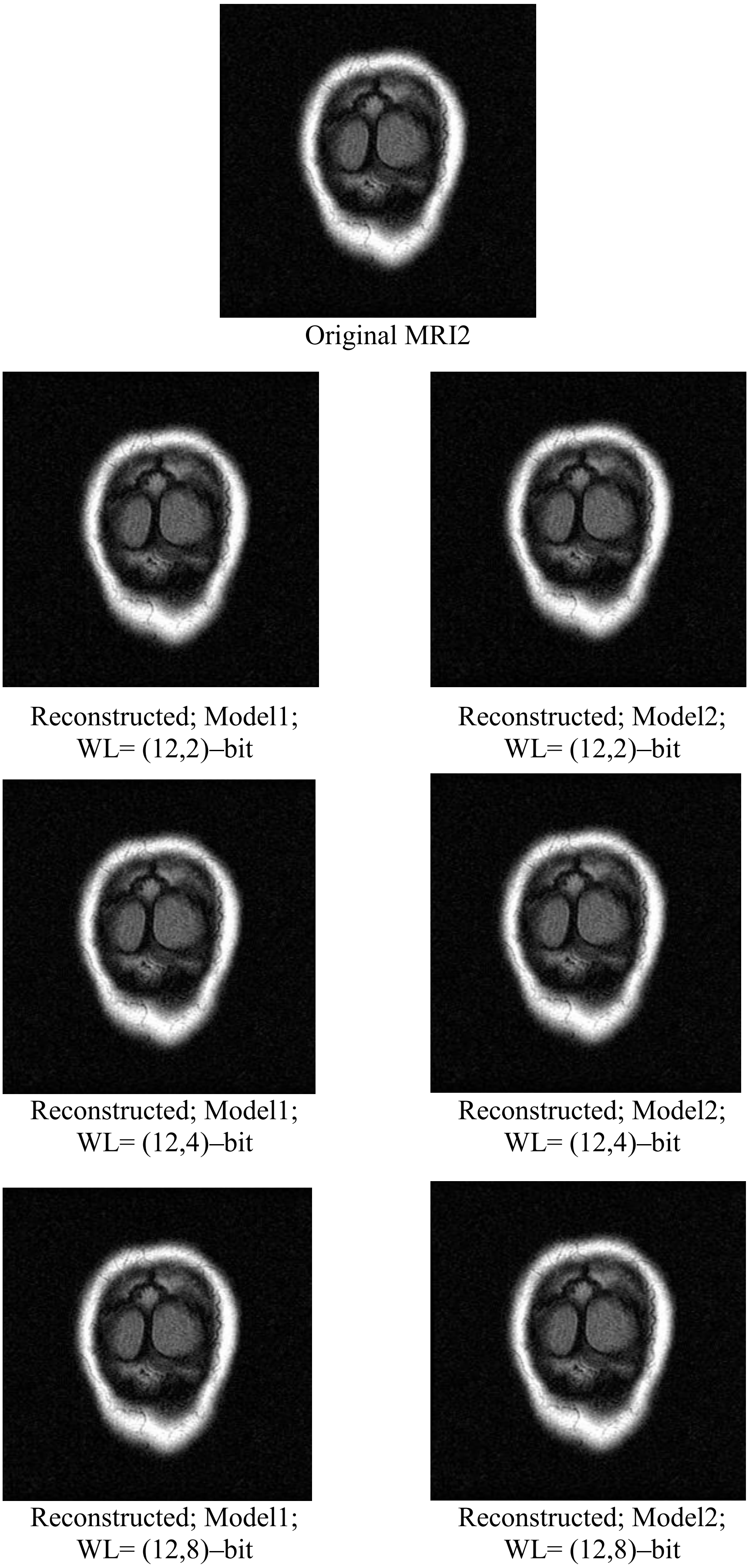
The PSNR and RMSE are used to check and evaluate the accuracy of the proposed architecture outputs and the reconstructed frames. These two metrics are computed between original and reconstructed frames. Further, the average of maximum absolute error of the 3-D DCT coefficients computed using architectures output and Matlab software implementation, are also computed using (4.29). The PSNR, RMSE and average maximum absolute error results for the first 8 frames from every input sequence are tabulated in Table 5.3 using 20, 16 and 14-bit wordlengths.

Table 5.3: The PSNR and RMSE between the original and reconstructed 8 frames and the maximum absolute error between the 3-D DCT coefficients computed using architectures output and Matlab.

	Reconstructed and original frames						3-D DCT Coefficients		
	PSNR (dB)			RMSE			Max. absolute error		
Input video	(12,8)	(12,4)	(12,2)	(12,8)	(12,4)	(12,2)	(12,8)	(12,4)	(12,2)
MRI1	∞	60	48	0	0.25	0.99	0.013	0.22	0.88
MRI2	∞	57	45	0	0.36	1.41	0.014	0.29	1.01
Akiyo	∞	59	47	0	0.27	1.15	0.013	0.25	1.10
Stefan	∞	57	45	0	0.35	1.40	0.014	0.27	0.99
Suzie	∞	57	45	0	0.35	1.39	0.013	0.24	0.91
Bus	∞	57	45	0	0.35	1.40	0.015	0.24	1.15
Flower	∞	57	45	0	0.35	1.39	0.015	0.25	1.01
Calendar	∞	57	45	0	0.35	1.40	0.013	0.22	0.93
Average	∞	58	46	0	0.33	1.32	0.0137	0.246	1.00

It is obvious from Table 5.3 that 100 % output accuracy is obtained using a 20-bit wordlength for all input sequences. Further, at 14-bit wordlength a PSNR of up to 48 dB and 0.9 lowest RMSE can be achieved.   Moreover, from Table 5.3, the proposed architectures produce very good image quality using the selected wordlengths. The average PSNR of the eight test sequences are ∞, 58 and 46 dB using (12, 8), (12, 4) and (12, 2)-bit wordlengths, respectively. The average RMSE between the original and the reconstructed frames is less than 1.3. While, the maximum absolute error between 3-D DCT coefficients computed using Matlab and the proposed architectures is less than 1.15.

Further, the first image of the original and reconstructed MRI2 medical image is shown in Figure 5.18 to give the reader a visual impression of the performance of the proposed architectures. In this figure, the reconstructed images are computed in both architectures using wordlength sizes of (12,2), (12,4) and (12,8)-bit. It is obvious that both architectures produce the same image quality. Also, it can be noticed that the (12,2)-bit wordlength produced a good image quality where the visual error cannot be recognised by the human visual system.



Original MRI2

Reconstructed; Model1; WL= (12,2)–bit

Reconstructed; Model2; WL= (12,2)–bit

Reconstructed; Model1; WL= (12,4)–bit

Reconstructed; Model2; WL= (12,4)–bit

Reconstructed; Model1; WL= (12,8)–bit

Reconstructed; Model2; WL= (12,8)–bit

Figure 5.18: The original and reconstructed MRI2 using Model1 and Model2 architectures for different wordlengths.

5.5.2 Hardware usage

The hardware usage, speed and computation time of both architectures using different output wordlengths is shown in Table 5.4. It is clear that the average hardware usage of Model1 and Model2 architectures are lower than 10 and 16 %, respectively. The hardware usage of Model2 has increased due to duplicating some circuits for post addition stages to perform an addition process on both data lines (first and second outputs). However, this extra hardware usage improves the maximum operating frequency of Model2 by more than 80 MHz over the Model1. As such, the minimum computation time of 512×512×8-pixel in Model2 is 6.4 ms, whereas 8.4 ms for Model1 is required, as shown in Table 5.4.

Table 5.4: Percentage of hardware usages, maximum operating frequencies and computation times of both models using different wordlengths.

			Model1			Model2		
			(12,8)	(12,4)	(12,2)	(12,8)	(12,4)	(12,2)
		Available	Rate %	Rate %	Rate %	Rate %	Rate %	Rate %
Hardware usage	Slice Registers	28,800	6	5	5	13	11	10
	LUTs	28,800	7	6	5	11	9	8
	Occupied Slices	7,200	10	7	7	17	13	13
	IOBs	480	6	5	5	6	5	5
	Block RAMs	60	15	15	15	20	18	16
	DSP48Es	48	18	16	16	33	25	25
	Average (%)		10	9	9	17	14	13
Maximum operating frequencies (MHz)		-	238	237	250	320	303	330
Computation times for 512×512×8-pixel data (ms)		-	8.8	8.9	8.4	6.6	6.9	6.4

An interesting point from Table 5.4 is that the memory requirements of both architectures are kept low due to the in-place computations in all butterfly and post addition stages. An additional reason behind that is the low memory requirement for the BRO and 3-D reordering operations. As such, a memory of $5N^2$ and $3N^2$-word has been used for 3-BRO for Model1 and Model2, respectively. Further, a memory of $5N^2$-word for 3-D reordering operation has been used in both models. Thus, the total number of block memory used in each model is less than 20% of the available memory resources of the 5vlx50tff1136-3 FPGA device (60 Block RAMs are available in this device).

5.5.3 Dynamic Power Consumption

The power consumption in FPGA is classified into static and dynamic power. The static power mainly comes from leakage current, whereas charging switch capacitors and short circuit currents are the main sources of dynamic power. The first type (static power) is affected by the technology used in FPGA manufacturing, whereas the second type (dynamic power) can be minimized by switching capacitance reduction, as in [136] and [137]. The dynamic power consumption of the proposed architectures is shown in Figure 5.19. The power consumption has been computed using a Xilinx Xpower analyser for various clock frequencies, and different wordlengths. The dynamic power consumption has increased in Model2 by around 15 mW over Model1; an important reason behind that is the additional multipliers in the butterfly stages and the duplication of some resources in the first two post addition stages. Thus, Model1 is outperforming Model2 in power consumption attribute, which makes it suitable for power economy applications. Further, using a wordlength of (12,2)-bit in both models has a good impact on power consumption reduction, as shown in Figure 5.19.

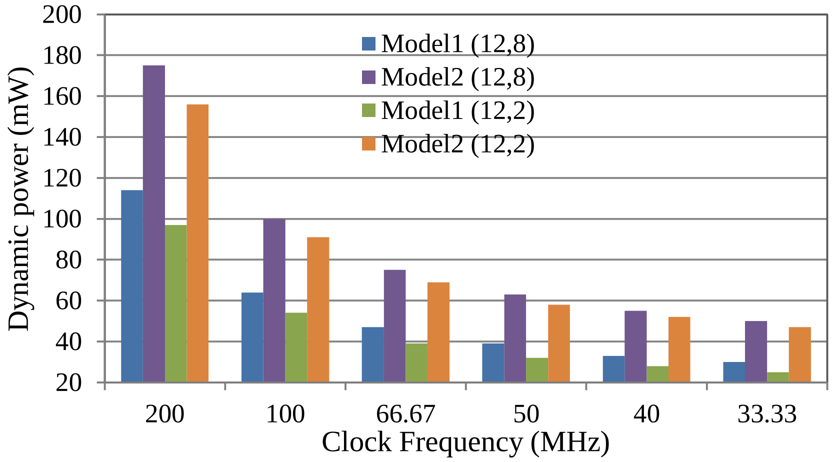


Figure 5.19: Dynamic power consumption of the proposed architectures.

5.5.4 Comparing with Other Architectures

The performance of the proposed architectures is compared with the performance of the architectures in [25] and chapter 4, as shown in Table 5.5 and Table 5.6. From Table 5.5, it is clear that the maximum operating frequency of the proposed architectures outperforms the normalised frequency of the 3-D DCT architectures in [25]. The normalised operating frequency is computed as in [25]. Furthermore, Model2 and 3-D DCT VR architecture in chapter 4 represent the highest speed DCT implementation among all architectures.

Further, from Table 5.6, it is clear that the hardware usage of the proposed architectures is less than that in [25]. An interesting point is that the number of multipliers in the proposed architectures is much less than those used in the block architecture in [25]. From Table 5.6, it can be noticed that Model1 architecture has the lowest number of registers and LUTs with respect to other architectures. While, the 3-D DCT RCF architecture has the advantage of the lowest number of DSP slices among all architectures. However, Model2 represents the faster architectures among all of them.

Table 5.5: Maximum clock frequency of the proposed and similar architectures.

	Proposed architectures				Normalised frequency of the architectures in [25]			
	This Chapter WL=20-bit		Chapter 4 WL= 21-bit					
	Model 1	Model 2	RCF	VR	Original	Pipelined ver.1	Pipelined ver.2	Block
Frequency (MHz)	250	330	237	259	132.82	152.63	132.82	175.73
Data cube size	8×8×8				8×8×8			
FPGA device	Xilinx Virtex5 5vlx50tff1136-3				Altera Cyclone II EP2C70F896C6			

Table 5.6: Hardware usage comparison between the 3-D DCT architectures.

Slice Logic Utilisation	This Chapter WL= 20-bit		Chapter 4 WL= 21-bit		Block architecture in [25]	
	Model1	Model2	RCF	VR		
Slice Registers	1,871	3,750	6,934	4,972	Registers	13,347
Slice LUTs	2,173	3,309	7,051	2,779	Logic Elements	19,355
Multipliers	7	10	39	4	9-bit Multipliers	128
Memory (Kb)	288	306	108	306	-	-
DSP48Es slices	9	16	0	6	-	-
Data cube size	8×8×8				8×8×8	
FPGA Device	XilinxVirtex5 5vlx50tff1136-3				Cyclone III EP3C120F780C8	

5.6 Summary

This chapter has introduced two new architectures for 3-D DCT computation using the 3-D DCT VR algorithm. The proposed architectures avoid the need for the memory for data transposition in the butterfly and post addition stages which in turn reduce the hardware usage and improve the processing speed. The proposed architectures are parameterisable in terms of wordlength, providing different output precision levels, power consumption, hardware usage and processing speeds. The proposed architectures have been implemented on Virtex 5 5vlx50tff1136-3 FPGA device and tested using different images and videos, different wordlengths and clock frequencies. The output accuracy, hardware usage, maximum operating frequencies and power consumption of the proposed architectures are evaluated using (12, 8), (12, 4) and (12, 2)-bit; (IWL, FWL) wordlengths. The accuracy tests revealed that the PSNR of the reconstructed images and frames are ∞, 60 and 47 dB using the three selected wordlengths, respectively. Also, the evaluated results reveal that the number of occupied slices is 722 and 1235 for the first and second architecture, respectively. Furthermore, the maximum operating frequencies achieved by the two architectures are 250 and 330 MHz using 14-bit output wordlength and 8×8×8-pixel input cube size for the first and second architectures, respectively. Moreover, the new architectures have the advantage of low power consumption, which is in the order of 25 to 175 mW, depending on the clock frequency and wordlength. Moreover, significant hardware usage reduction with higher operating frequencies is achieved when compared with similar 3-D DCT hardware architectures.

So far, block based multidimensional transforms for image and video processing have been discussed and different architectures have been proposed and verified. The next chapter will discuss a multidimensional discrete wavelet transform (DWT) which represents a non-block transformation for image and video processing.

Chapter 6: Parallel and Multiplierless Multidimensional CDF 9/7 DWT Architectures

6.1 Introduction

Thus far, block based multidimensional transforms as used in image and video compression applications have been discussed, with different architectures proposed and verified. This chapter will discuss the multidimensional DWT, which represents a non-block transform used in image and video processing applications. The DWT is used in audio, image and video compression algorithms as it overcomes the blocking artefact limitations of the conventional block-based image compression techniques. However, in spite of the advantages of the DWT over DCT and FFT, it has the limitations of high memory requirements and high computation costs. These factors affect the speed, complexity and power consumption of any DWT architecture, thus motivating many researchers to investigate and propose different versions of DWT computation approaches to overcome these limitations and drawbacks.

In this chapter, new lifting-based 1-D, 2-D and 3-D Cohen-Daubechies-Feauveau 9/7 (CDF 9/7) DWT parallel and multiplierless architectures are proposed. The proposed architectures have been devised to process an infinite GOF and achieve a good compromise between memory requirement and power consumption. Such architectures partition the data into blocks of 4, 4×4 and 4×4×4 pixels, which are processed concurrently. The temporal computation in the proposed 3-D DWT architecture is carried out using a block memory size of four frames only. Furthermore, shift-add multipliers with ignorable error are suggested to reduce the power consumption arising from using the constant multipliers. The evaluation results have revealed that the proposed architectures outperform the works in the literature in terms of processing speed and power consumption. In addition, further wavelet filters can be implemented using the proposed design methodology and architectures of the CDF 9/7 when appropriate modifications are introduced.

The remainder of this chapter is organised as follows. Section 6.2 reviews the lifting-based 1-D CDF 9/7 DWT. A modified CDF 9/7 lifting scheme is introduced in section 6.3. A 1-D DWT architecture that implements such modification is presented in section 6.4. Sections 6.5 and 6.6 and present new 2-D and 3-D CDF 9/7 DWT architectures. The results are analysed in section 6.7, while the summary is drawn in section 6.8.

6.2 The CDF 9/7 1-D DWT Lifting Scheme

The lifting-based DWT scheme decomposes a wavelet filter into a matrix product involving a sequence of upper and lower triangular matrices pairs and diagonal scaling matrix. Such an operation corresponds to factorisation of the poly-phase matrix of a target wavelet filter [80, 81]. The poly-phase decomposition can be expressed as follows:

$$h(z) = h_e(z^2) + z^{-1}h_0(z^2) \tag{6.1}$$

$$g(z) = g_e(z^2) + z^{-1}g_0(z^2) \tag{6.2}$$

$$P(z) = \begin{bmatrix} h_e(z) & g_e(z) \\ h_o(z) & g_o(z) \end{bmatrix} \tag{6.3}$$

$$P(z) = \left[\prod_{i=1}^{m}\left(\begin{bmatrix} 1 & s_i(z) \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ t_i(z) & 1 \end{bmatrix}\right)\right]\begin{bmatrix} k_0 & 0 \\ 0 & k_1 \end{bmatrix} \tag{6.4}$$

where $h(z)$ and $g(z)$ are the low and high pass analysis filters, respectively. $P(z)$ is the poly-phase decomposition matrix. As an example, the CDF 9/7 filter can be decomposed into four lifting and one scaling steps, as follows [28, 80, 81, 138]:

$$P(z) = \begin{bmatrix} 1 & \alpha(1+z^{-1}) \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ \beta(1+z) & 1 \end{bmatrix}\begin{bmatrix} 1 & \gamma(1+z^{-1}) \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ \delta(1+z) & 1 \end{bmatrix}\begin{bmatrix} \zeta & 0 \\ 0 & {}^{1}/_{\zeta} \end{bmatrix} \tag{6.5}$$

where α = −1.586134342, β = −0.05298011854, γ = 0.8829110762, δ = 0.4435068522, and ζ = 1.149604398.

Then, the computation steps can be implemented as follows [81]:

$$\begin{gathered} s_l^0 = x_{2l} \\ d_l^0 = x_{2l+1} \end{gathered} \tag{6.6}$$

$$d_l^1 = d_l^0 + \alpha(s_l^0 + s_{l+1}^0) \tag{6.7}$$

$$s_l^1 = s_l^0 + \beta(d_l^1 + d_{l-1}^1)$$

$$d_l^2 = d_l^1 + \gamma\,(s_l^1 + s_{l+1}^1)$$
$$s_l^2 = s_l^1 + \delta\,(d_l^2 + d_{l-1}^2) \tag{6.8}$$

$$s_l = \zeta s_l^2$$
$$d_l = {d_l^2}/{\zeta} \tag{6.9}$$

where (6.6) represents splitting of signal x into odd and even indexed terms; it is also termed the lazy wavelet step, $d_l^1\ and\ d_l^2$ are the first and second predict steps, and $s_l^1\ and\ s_l^2$ are the first and second update steps, while (6.9) represents the final scaling step. The computation of (6.9) as a whole involves a series of five multiplication and eight addition operations. As such, the hardware implementation of (6.9) results in a critical path of 5Tm + 8Ta; where Tm and Ta are the multiplier and adder delay, respectively [28, 139, 140]. Such critical delay can be shortened to 3Tm + 4Ta by adopting a flipping structure [139, 140]. This modified structure is based on the modification of (6.6) to (6.9) as follows:

$$s_l^0 = x_{2l}$$
$$d_l^0 = x_{2l+1} \tag{6.10}$$

$$d_l^1 = A \times d_l^0 + (s_l^0 + s_{l+1}^0)$$
$$s_l^1 = B \times s_l^0 + \frac{1}{16}(d_l^1 + d_{l-1}^1) \tag{6.11}$$

$$d_l^2 = C \times d_l^1 + \frac{1}{2}\,(s_l^1 + s_{l+1}^1)$$
$$s_l^2 = D \times s_l^1 + \frac{1}{2}\,(d_l^2 + d_{l-1}^2) \tag{6.12}$$

$$s_l = K_0 s_l^2$$
$$d_l = K_1 d_l^2 \tag{6.13}$$

The constants $A, B, C\ and\ D$ can be calculated as follows: $A = {}^{1}/_{\alpha} = -0.6304636$, $B = {}^{1}/_{16\alpha\beta} = 0.743750255$, $C = {}^{1}/_{32\gamma\beta} = -0.669067178$ and $D = {}^{1}/_{4\gamma\delta} = 0.638443853$. The scaling factors $(K_0\ and\ K_1)$ can be computed using: $K_0 = 64\alpha\beta\gamma\delta = 2.42102112$ and $K_1 = {}^{32\alpha\beta\gamma}/_{\delta} = 2.06524422$ [140].

6.3 The Proposed Lifting-based CDF 9/7 DWT Computation Scheme

The computation scheme of (6.10) to (6.13) is shown in Figure 6.1. In this figure, A, B, C, D, K_0 and K_1 are the constant and scaling factors. From this figure, nine input samples are required to compute the wavelet coefficients s_2 and d_2.

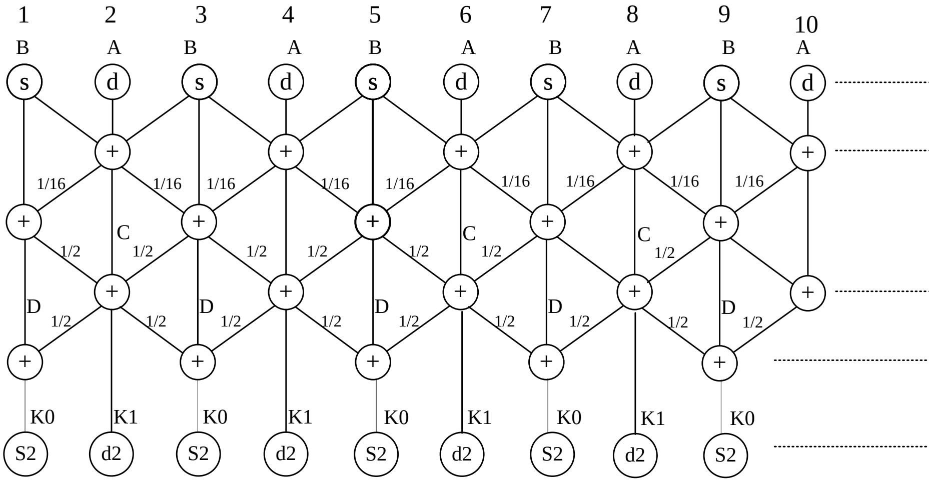


Figure 6.1: The CDF 9/7 1-D DWT [28].

The proposed 1-D CDF 9/7 DWT computation scheme is shown in Figure 6.2. In this scheme, the input data sequence $\{x_0, x_1, \dots, x_{P-1}\}$; P is the input data length, is partitioned into a blocks of four samples, and thus the first block is $x_{00} = \{x_0, x_1, x_2, x_3\}$. Such samples are termed $\{s_1^0, d_1^0, s_2^0, d_2^0\}$ or even and odd samples and fed to the 1-D DWT computation unit in parallel. During the computation process of the first data block - the first computation round - four temporary values are computed and allocated in appropriate registers, while the output wavelet coefficients are set to zero. The four temporary values, marked by the shaded left skew bar shown in Figure 6.2, are: $\widetilde{s_1^2}, \widetilde{d_1^2}, \widetilde{s_2^1}\ and\ \widetilde{d_2^1}$ and they are computed according to the following sequence:

$$d_1^1 = A \times d_1^0 + (s_1^0 + s_2^0) \tag{6.14}$$

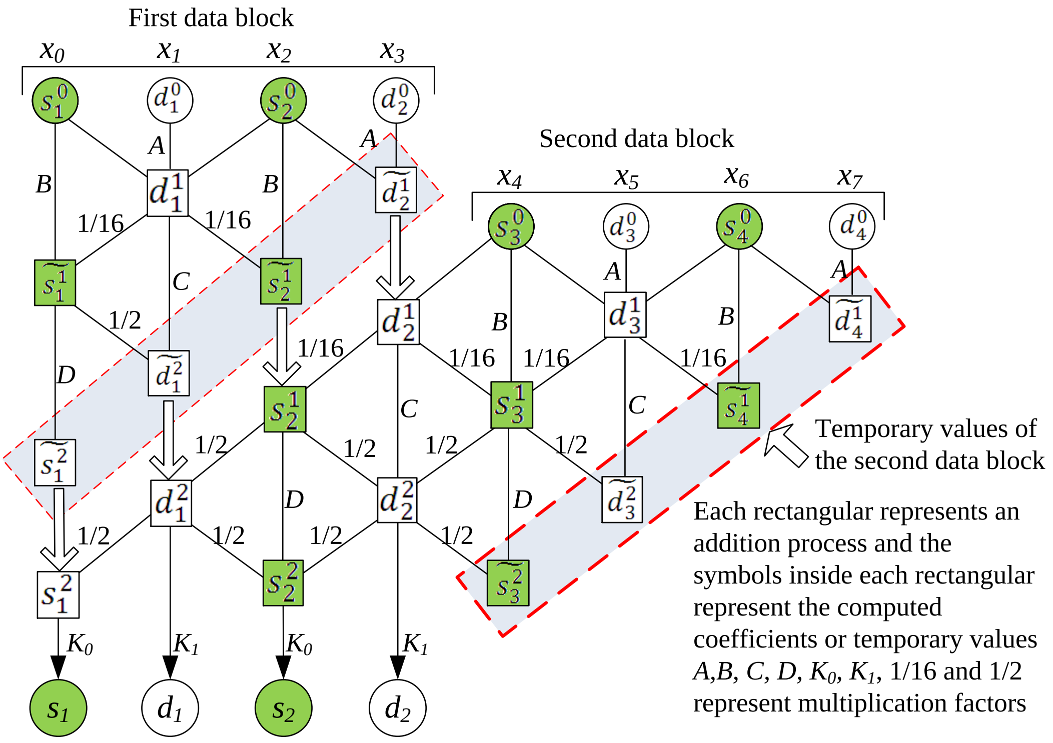
$$\widetilde{d_2^1} = A \times d_2^0 + (s_2^0 + 0) \tag{6.15}$$

$$\widetilde{s_1^1} = B \times s_1^0 + \frac{1}{16}(d_1^1 + 0) \tag{6.16}$$

$$\widetilde{s_2^1} = B \times s_2^0 + \frac{1}{16}(d_1^1 + 0) \tag{6.17}$$

$$\widetilde{d_1^2} = C \times d_1^1 + \frac{1}{2}\left(\widetilde{s_1^1} + 0\right) \tag{6.18}$$

$$\widetilde{s_1^2} = D \times \widetilde{s_1^1} + \frac{1}{2}\ (0 + 0) \tag{6.19}$$



Temporary values of the second data block

Each rectangular represents an addition process and the symbols inside each rectangular represent the computed coefficients or temporary values *A*,*B*, *C*, *D*, *K₀*, *K₁*, 1/16 and 1/2 represent multiplication factors

Figure 6.2: The proposed CDF 9/7 computation scheme for the first two data blocks.

During the second data block, the second computation round of $x_{01} = \{x_4, x_5, x_6, x_7\}$ $(\{s_3^0, d_3^0, s_4^0, d_4^0\})$, four output wavelet coefficients are computed using the temporary values and the new input block samples as described in the following computation sequence:

$$d_2^1 = s_3^0 + \widetilde{d_2^1} \tag{6.20}$$

$$d_3^1 = A \times d_3^0 + (s_3^0 + s_4^0) \tag{6.21}$$

$$\widetilde{d_4^1} = A \times d_4^0 + (s_4^0 + 0) \tag{6.22}$$

$$s_2^1 = \widetilde{s_2^1} + \frac{1}{16} d_2^1 \tag{6.23}$$

$$s_3^1 = B \times s_3^0 + \frac{1}{16}(d_2^1 + d_3^1) \tag{6.24}$$

$$\widetilde{s_4^1} = B \times s_4^0 + \frac{1}{16}(d_3^1 + 0) \tag{6.25}$$

$$d_1^2 = \widetilde{d_1^2} + \frac{1}{2} s_2^1 \tag{6.26}$$

$$d_2^2 = C \times d_2^1 + \frac{1}{2}(s_2^1 + s_3^1) \tag{6.27}$$

$$\widetilde{d_3^2} = C \times d_3^1 + \frac{1}{2}(s_3^1 + 0) \tag{6.28}$$

$$s_1^2 = \widetilde{s_1^2} + \frac{1}{2} d_1^2 \tag{6.29}$$

$$s_2^2 = D \times s_2^1 + \frac{1}{2}(d_2^2 + d_1^2) \tag{6.30}$$

$$\widetilde{s_3^2} = D \times s_3^1 + \frac{1}{2}(d_2^2 + 0) \tag{6.31}$$

Then, the four wavelet coefficients ($s_1, d_1,\ s_2$ and d_2) are computed as follows:

$$s_1 = K_0 \times s_1^2 \tag{6.32}$$

$$d_1 = K_1 \times d_1^2 \tag{6.33}$$

$$s_2 = K_0 \times s_2^2 \tag{6.34}$$

$$d_2 = K_1 \times d_2^2 \tag{6.35}$$

Furthermore, four new temporary values are computed and allocated in the same four registers to be used during the computation process of the third input block. The second four temporary values are $\widetilde{s_3^2}, \widetilde{d_3^2}, \widetilde{s_4^1}$ and $\widetilde{d_4^1}$ and these are computed using (6.31), (6.28), (6.25) and (6.22), respectively. This process is repeated ${}^{P}\!/_{4}$ times until the wavelet coefficients of the last input block are computed. The abovementioned computation procedure represents full 1-D wavelet computation of P-point 1-D input data. Thus, in the proposed 1-D DWT scheme four temporary values with four new data samples can be used to evaluate the DWT coefficients. The proposed computation scheme can be used to perform the 2-D and 3-D CDF 9/7 DWT computation. In the proposed 3-D DWT computation scheme a memory depth of four frames is used to perform the computation operations without any computational redundancy.

6.4 The Proposed Lifting-based 1-D DWT Architecture

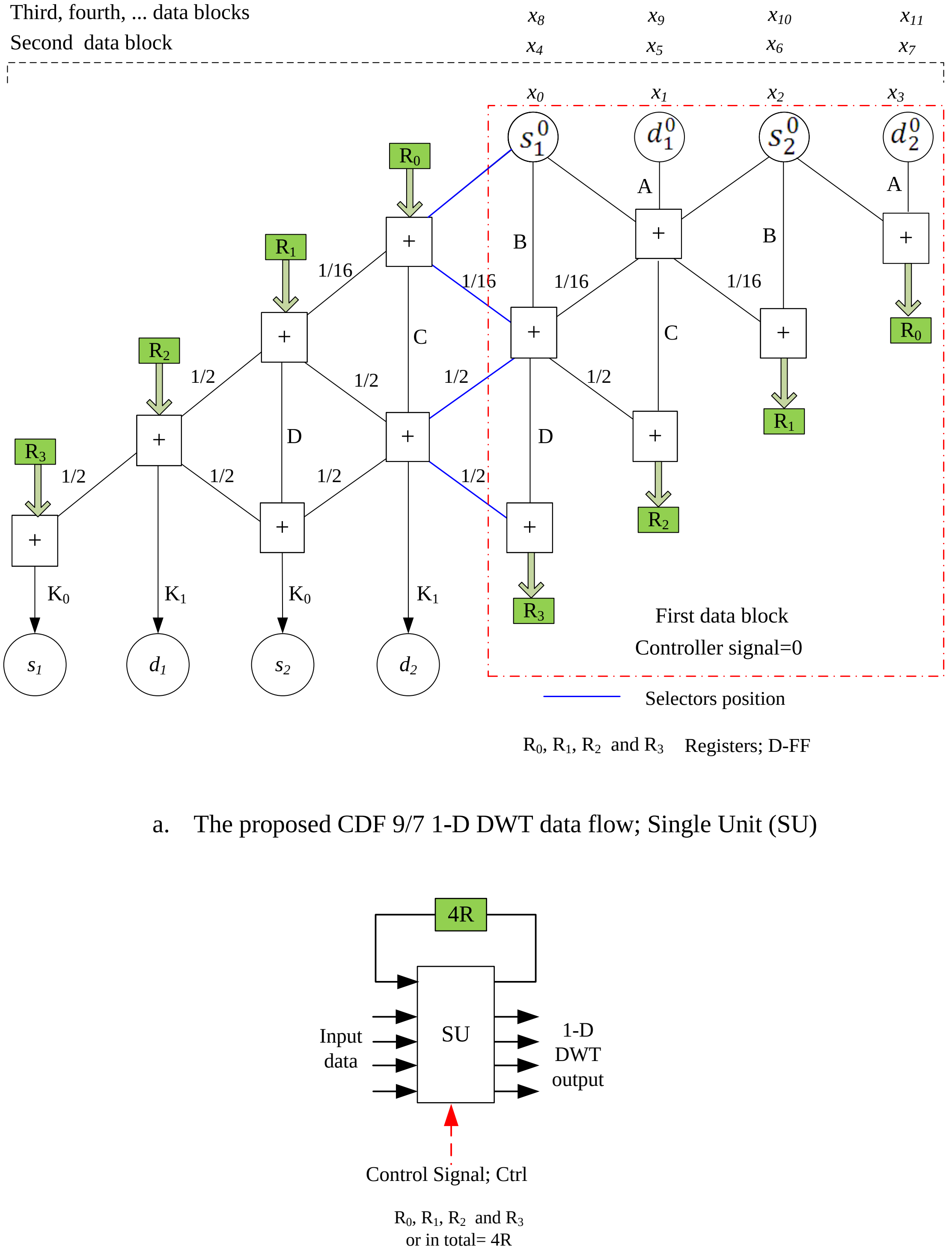
The proposed lifting-based CDF 9/7 1-D DWT architecture is based on the computation process described in section 6.3, as shown in Figure 6.3-a. In this figure, the parameters R_0, R_1, R_2 and R_3 represent the registers used to store the temporary values during the computation process. R_0, R_1, R_2 and R_3 all have the same length; thus, they have been substituted by 4R in Figure 6.3-b. In this figure, the blue lines represent the position of the selectors or switches. The *Ctrl* signal represents the Controller signal.

In the rest of this chapter, each 1-D DWT lifting architecture unit is termed Single Unit (SU). The proposed SU, represents the CDF 9/7 1-D DWT architecture. The word data block refers to 4 and 4×4-sample in the proposed 1-D and 2-D DWT architectures, respectively. Each SU has four input and four output ports for data signals, as well as one input port for the controller signal.

The computation procedure through the SU is accomplished according to the following sequence: a P-sample 1-D input data is partitioned into blocks of 4-sample which is fed to the SU in parallel. As an example, an input data x of P samples is partitioned into $P/4$ data blocks, as follows:

$$\{x(4i), x(4i+1), x(4i+2), x(4i+3)\}, i = 0, 1, 2, \dots \frac{P}{4} - 1 \qquad (6.36)$$

Hence, during the first data block, $i = 0$, the SU, is set to perform the computation of (6.14) to (6.19) according to the controller signal. The controller signal is 0 and all selectors are disconnected during the first block computation period; the controller unit and its signals is shown the Appendix F. Throughout this period; $i = 0$, four temporary values are computed and allocated in four registers. Such temporary values are used during the computation operation of the subsequent data block; $i = 1$. Then, the selectors are reconnected during the computation process of the blocks $i = 1$ to $i = \frac{P}{4} - 1$. Four wavelet coefficients are computed during the computation operation of every input data block. These four wavelet coefficients are arranged in the following sequence: Low, High, Low and High pass (L, H, L and H) coefficients. This process is repeated P/4 times and, during each input data block, four temporary values are allocated in the same four registers and four wavelet coefficients are computed.



a. The proposed CDF 9/7 1-D DWT data flow; Single Unit (SU)

b. The block diagram of SU

Figure 6.3: Single unit (SU) of the CDF 9/7 1-D DWT architecture.

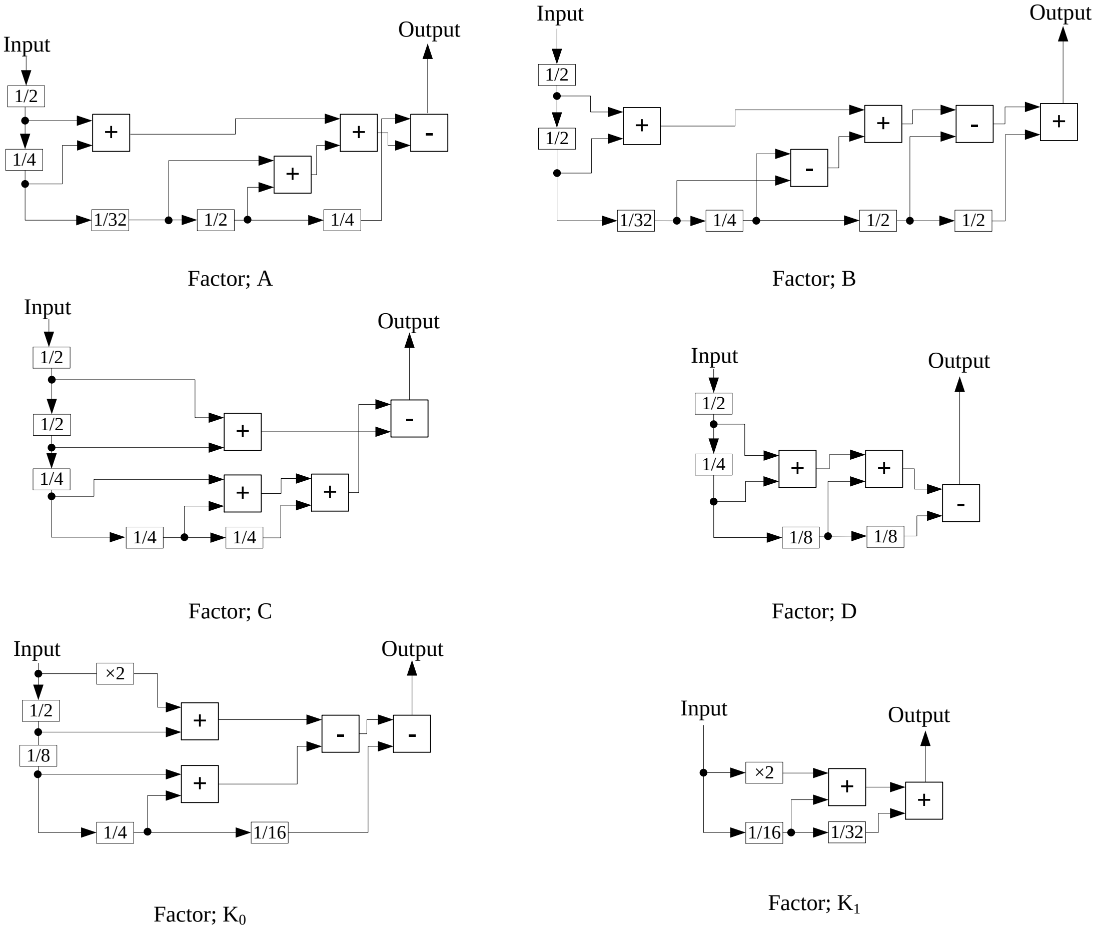
The proposed SU architecture is designed to tackle different wordlengths to provide various levels of output precision. The wordlength of each SU is partitioned into two parts: the IWL and FWL. In the rest of this chapter, the IWL is set to an 11-bit signed integer and the FWL is selected to be 5 or 7; however, any other wordlength sizes can be chosen as required.

The proposed SU involves 12 constant multipliers to perform the multiplication operation with the previously mentioned constants and scaling factors; A, B, C, D, K_0 and K_1. However, it is well-known that the multiplication operation affects the power consumption of any system. Thus, each multiplier is substituted by a proposed shift-add element, as shown in Figure 6.4. In this figure, the division or multiplication by a power of two values is substituted by shift operations.

As shown in Table 6.1, the proposed shift-add multipliers are designed to produce the same output accuracy of the conventional multipliers. However, a negligible error between each two multipliers occurred, as specified in Table 6.1. From this table, it is obvious that the maximum error (difference) between the proposed shift-add and conventional multipliers is less than 0.79×10^{-3}; thus, it can be neglected.

Table 6.1: The difference between the constant and proposed shift-add multipliers.

Constant	Conventional multipliers	Proposed shift-add multipliers	difference
A	0.630463621	0.630371094	9.25272×10^{-5}
B	0.743750255	0.743652344	9.79113×10^{-5}
C	0.668067178	0.66796875	9.8428×10^{-5}
D	0.638443853	0.638671875	-2.28022×10^{-4}
K_0	2.421021123	2.420898438	1.22686×10^{-4}
K_1	2.065244222	2.064453125	7.91097×10^{-4}



Factor; A

Factor; B

Factor; C

Factor; D

Factor; K_0

Factor; K_1

Figure 6.4: The proposed shift-add multipliers.

6.5 The Proposed Lifting-based 2-D DWT Parallel Architecture

The proposed 2-D DWT architecture consists of four SUs for rows (RUs) and four SUs for columns (CUs) computation units, as shown in Figure 6.5. The SelR and SelC signals are used to control the transitions between the block boundaries.

Initially, each input image is partitioned into blocks of 4×4-pixel. Each row is fed to a corresponding RU. Thus, an input image with P × Q-pixel is partitioned into $\frac{P\times Q}{16}$ data blocks using the suggested 4×4-pixel data blocks. Such blocks are shown in Figure 6.6 for an image tile of 8×8-pixel.

The 16-sample data stream is converted into four parallel data sequences using a serial to parallel converter. Each four sample block belongs to a single row from the original 4×4-pixel data block. Thus, four 4-sample blocks are fed to four RUs in parallel. The input data to RU1, RU2, RU3 and RU4 are as follows:

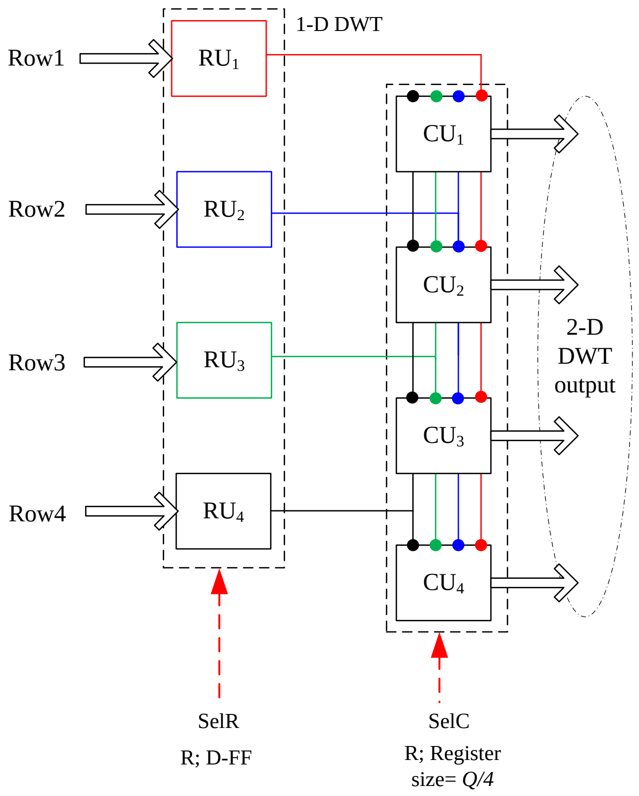
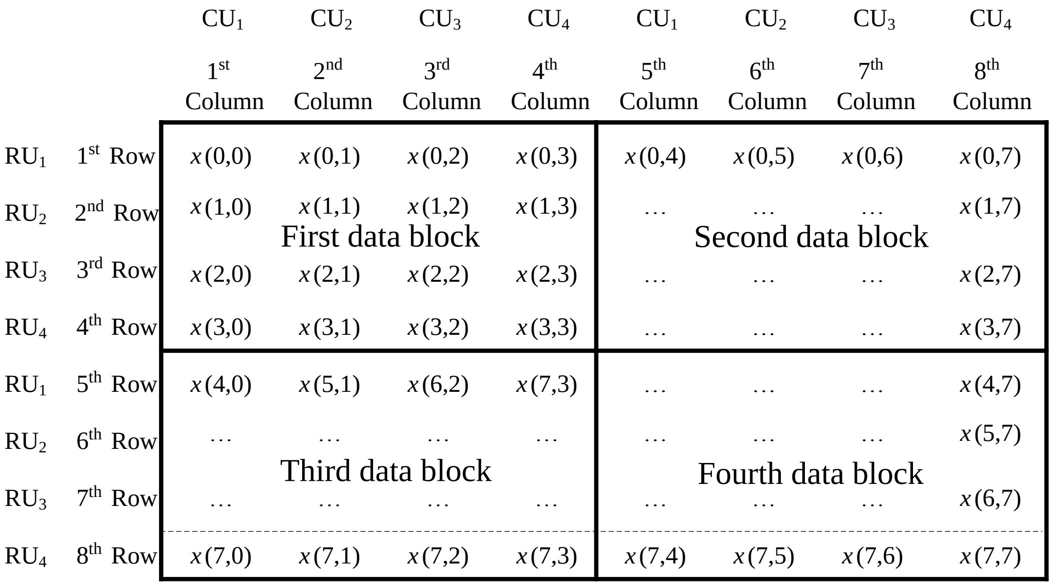


Figure 6.5 : The proposed 2-D DWT parallel architecture.



		CU$_1$ 1st Column	CU$_2$ 2nd Column	CU$_3$ 3rd Column	CU$_4$ 4th Column	CU$_1$ 5th Column	CU$_2$ 6th Column	CU$_3$ 7th Column	CU$_4$ 8th Column
RU$_1$	1st Row	$x(0,0)$	$x(0,1)$	$x(0,2)$	$x(0,3)$	$x(0,4)$	$x(0,5)$	$x(0,6)$	$x(0,7)$
RU$_2$	2nd Row	$x(1,0)$	$x(1,1)$	$x(1,2)$	$x(1,3)$	…	…	…	$x(1,7)$
			First data block				Second data block		
RU$_3$	3rd Row	$x(2,0)$	$x(2,1)$	$x(2,2)$	$x(2,3)$	…	…	…	$x(2,7)$
RU$_4$	4th Row	$x(3,0)$	$x(3,1)$	$x(3,2)$	$x(3,3)$	…	…	…	$x(3,7)$
RU$_1$	5th Row	$x(4,0)$	$x(5,1)$	$x(6,2)$	$x(7,3)$	…	…	…	$x(4,7)$
RU$_2$	6th Row	…	…	…	…	…	…	…	$x(5,7)$
			Third data block				Fourth data block		
RU$_3$	7th Row	…	…	…	…	…	…	…	$x(6,7)$
RU$_4$	8th Row	$x(7,0)$	$x(7,1)$	$x(7,2)$	$x(7,3)$	$x(7,4)$	$x(7,5)$	$x(7,6)$	$x(7,7)$

Figure 6.6: Four 4×4-sample data blocks.

$$\{x(4i,4j), x(4i,4j+1), x(4i,4j+2), x(4i,4j+3)\} \tag{6.37}$$

$$\{x(4i+1,4j), x(4i+1,4j+1), x(4i+1,4j+2), x(4i+1,4j+3)\} \tag{6.38}$$

$$\{x(4i+2,4j), x(4i+2,4j+1), x(4i+2,4j+2), x(4i+2,4j+3)\} \tag{6.39}$$

$$\{x(4i+3,4j), x(4i+3,4j+1), x(4i+3,4j+2), x(4i+3,4j+3)\} \tag{6.40}$$

where $i = 0,1,2,\ldots\frac{P}{4} - 1$ and $j = 0,1,2,\ldots\frac{Q}{4} - 1$ represent block row and column indices, respectively.

Consequently, during the first 4×4-pixel block ($i = 0\, and\, j = 0$), each RU computes four temporary values $\widetilde{s_1^2}, \widetilde{d_1^2}, \widetilde{s_2^1}\ and\ \widetilde{d_2^1}$. These temporary values are stored in the four registers. Each register has a single stage length; R = 1, as shown in Figure 6.3, and the row controlling signal; Row Selector (SelR) is set to 0; the controller unit is detailed in Appendix F. The four temporary values will be used during the computation process of the subsequent data block. Hence, when $i = 0\, and\, j = 1$ (the second data block), the SelR is set to 1 and four wavelet coefficients are computed from each RU; 2 coarse and 2 fine components. At the same time, the content of the four registers in each RU is updated with new temporary values to be ready for the third data block. The first four outputs of each RU become available for the columns filtering process after 161 clock cycles regardless of the image size. The above mentioned procedure represents the row wavelet filtering step. Subsequently, the first four outputs of each RU become available for the columns filtering process after 161 clock cycles regardless of the image size

Then, the subsequent units (CUs), perform the filtering operation on the output of RUs. This step represents the column wavelet filtering operation. It starts the computation process when each RU has finished the computation process of the first data block. During the first four rows of the input image ($i = 0$ and $j \geq 0$) each CU computes $Q/4$ temporary values which are allocated in a dedicated chain of registers with a length of $Q/4$; $R = Q/4$, as shown in Figure 6.5. These temporary values will be used during the computation process of the subsequent four rows (rows 5 to 8; $i = 1\ and\ j \geq 0$). During the period of rows 5 to 8 computation process, each CU computes four wavelet coefficients using the temporary values and the new RUs outputs. New temporary values are also allocated in the same registers to be used during the computation operation of the ensuing four rows (rows 9 to 12; $i = 2\ and\ j \geq 0$. Consequently, the outputs from this step (CUs outputs) are arranged in the sequence of Low-Low, Low-

High, High-Low and High-High pass filtered signals (LL, LH, HL and HH) wavelet bands which represent the 2-D wavelet coefficients of the input image. The first four 2-D wavelet coefficients become available at the output ports after $4(Q + 73) + 1$ clock cycles.

6.6 The Proposed Lifting-based 3-D DWT Parallel Architecture

This section presents CDF 9/7 3-D DWT lifting-based parallel architecture based on the proposed SU. In this architecture, the 3-D DWT computation is performed by applying 1-D DWT on the rows, columns and frames, successively. It consists of; four Row-Column units (RCUs) and four SUs for frame direction (FUs), as shown in Figure 6.7. Each RCU calculates 2-D DWT coefficients for a single frame and the outputs of these units are fed to four FUs. The four FUs are termed; FU_1, FU_2, FU_3 and FU_4, and these are used to compute the wavelet coefficients in the frame direction.

The 3-D input signal is considered to have P-Rows, Q-Columns and F-Frames; $P \times Q \times F$-pixel and each dimension is assumed to be a multiple of 4 in all subsequent explanations throughout this chapter.

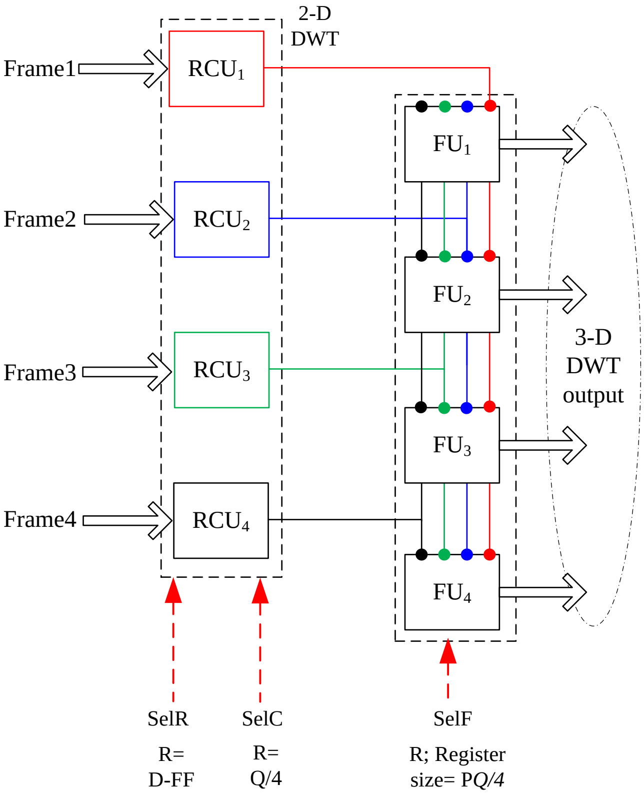


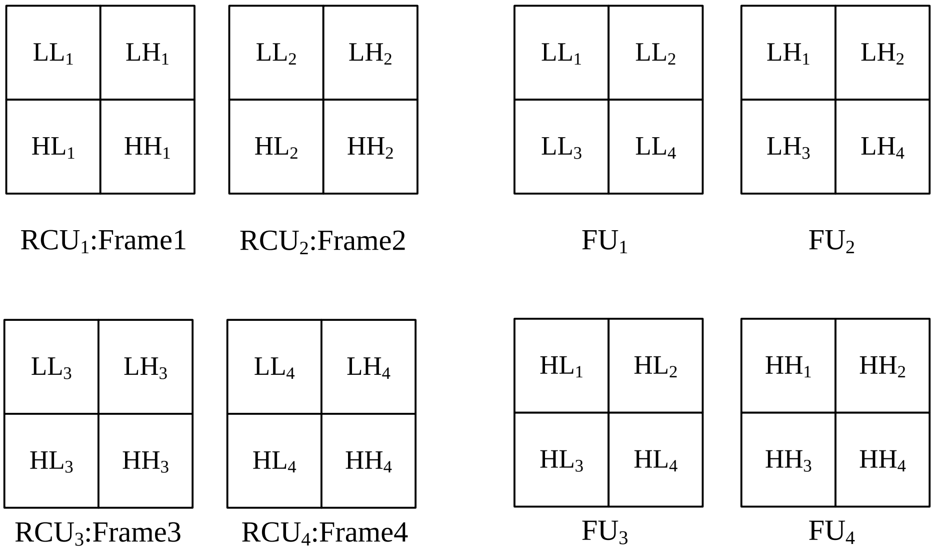
Figure 6.7: Block diagram of the proposed 3-D DWT lifting-based architecture.

6.6.1 Row-Column Units (RCUs)

The input signal of the proposed 3-D DWT architecture is partitioned into a group of four frames (GOF) : $F_{4n}, F_{4n+1}, F_{4n+2}$ and F_{4n+3}, $n = 0, 1, 2, \dots, \frac{F}{4}$, where each frame is fed to a single RCU, concurrently. The operation of the four RCUs is similar to the operation of the proposed 2-D DWT architecture. The RCUs are identical and operate simultaneously to compute the 2-D wavelet coefficients of each input frame. Thus, each RCU produces LL, LH, HL and HH wavelet bands for each input frame. These outputs are termed: LL_m, LH_m, HL_m and HH_m where, $m = 1,2,3,4$ and represents the RCUs index.

6.6.2 Frame Units (FUs)

This stage computes wavelet coefficients in the frame direction using four FUs. These units are similar to that shown in Figure 6.3 with a different controlling signal and buffer size (register R); the controller unit is detailed in Appendix F. The inputs to this stage are collected from the outputs of the RCUs. Hence, the inputs to the FU1 are the four LL bands which can be collected from the four RCUs, and likewise for LH, HL and HH bands, as shown in Figure 6.8. This figure explains how the output wavelet bands of each frame are fed from RCUs to FUs.


LL_1 LH_1 HL_1 HH_1
RCU_1:Frame1
LL_2 LH_2 HL_2 HH_2
RCU_2:Frame2
LL_3 LH_3 HL_3 HH_3
RCU_3:Frame3
LL_4 LH_4 HL_4 HH_4
RCU_4:Frame4
LL_1 LL_2 LL_3 LL_4
FU_1
LH_1 LH_2 LH_3 LH_4
FU_2
HL_1 HL_2 HL_3 HL_4
FU_3
HH_1 HH_2 HH_3 HH_4
FU_4

a. Row-Column units outputs b. Frame units inputs

Figure 6.8: Input bands to the frame units.

During the first four frames (first computation round; first GOF),$\mathrm{F}_0, \mathrm{F}_1, \mathrm{F}_2$ and $\mathrm{F}_3, \mathrm{n} = 0$, each FU computes and stores four temporary values every clock cycle to perform the computation operations of (6.14) to (6.19). The computed temporary values are allocated in four block RAMs. Each RAM has a depth of $R = \frac{PQ}{4}$ words, as shown in Figure 6.7, and represents a temporal buffer. Thus, the total memory depth required for each FU is $4\left(\frac{\mathrm{PQ}}{4}\right)$-word (four R-registers). These block memories act as a long delay line which can be constructed using a single port block RAM , dual port block RAM, FIFO or long chain of registers. Thus, the total block temporal memory requirement for the proposed 3-D DWT architectures is: $16\left(\frac{\mathrm{PQ}}{4}\right)$, which represents the memory required by all FUs which are sufficient to handle four frames as temporary values (single GOF). In the subsequent computation round, when $\mathrm{n} = 1$, the input frames are; $\mathrm{F}_4, \mathrm{F}_5, \mathrm{F}_6$ and F_7, and again the RCUs compute the 2-D DWT coefficients for the new frames. These outputs are also fed to the FUs in the same sequence as the first GOF. Hence, throughout the second computation round ($\mathrm{n} = 1$), each FU computes the 3-D DWT coefficient using the RCU outputs and the temporary values allocated in the frame buffers. Furthermore, at each computation round, the temporary values in each block memory are updated with new values to be used in the subsequent computation round. The temporary values are stored and recalled in a sequential order; thus, FIFOs or chain of registers can be used without the need for additional addressing circuits. Moreover, the processing time for any number of frames and frame sizes is $PF\frac{Q}{4}$ clock cycles. Furthermore, the proposed 3-D DWT architecture is constructed using the same basic unit (SU) for rows, columns and frame directions and it is parameterisable in terms of wordlength, frame sizes and number of frames. These parameters are IWL, FWL, P, Q and F, respectively.

6.7 Results and Discussion

In this section, the performance of the proposed 2-D and 3-D DWT architectures are analysed. The proposed architectures have been implemented and tested using a 6vlx760ff1760-2Virtex 6 FPGA device. The output accuracy, power consumption and hardware cost have been analysed using various images and video sequences, as well as different wordlengths. These images and video sequences are shown in Figure 3.15 and Figure 4.19. Further, the proposed architectures have been designed and implemented using Xilinx System generator tool, as illustrated in section 2.8. The initialisation Matlab codes for the proposed architectures are listed in the Appendix G.

6.7.1 The 2-D DWT Architecture Performance

6.7.1.1 The Rate Distortion Performance

The PSNR between the original and the reconstructed images are computed and tabulated in Table 6.2. The maximum 2-D DWT absolute error between the proposed architecture and its Matlab implementation are also shown in this table. The wordlengths used in these tests are 18 and 16-bit, where the IWL is set to 11-bit with 7 and 5-bit for FWL. However, the architecture is designed to tackle any wordlengths as specified or required, according to the target accuracy and PSNR. As shown in Table 6.2, the average of the maximum absolute error in wavelet coefficients of the proposed architecture using the MRI images and video sequences are 0.32 and 1.32 with PSNRs of ∞ and 53 dB, respectively. This error occurs due to the rounding and truncation operations of the proposed shift-add constant multipliers. However, it does not have a high impact on the targeted PSNR of the reconstructed frames.

Table 6.2: The PSNR and the maximum absolute error for the proposed 2-D DWT architecture using 18-bit and 16-bit WL.

WL (IWL, FWL)		(11, 7)		(11, 5)	
Images	Size (P×Q)	PSNR (dB)	Max. absolute Error	PSNR (dB)	Max. absolute Error
MRI1	128×128	∞	0.30	57	1.19
MRI2	256×256	∞	0.36	52	1.34
Akiyo	144×176	∞	0.33	52	1.38
Stefan	288×352	∞	0.30	52	1.40
Suzie	144×176	∞	0.30	52	1.26
Bus	288×352	∞	0.32	52	1.34
Flower	288×352	∞	0.31	52	1.35
Calendar	144×176	∞	0.32	52	1.28
Lena	512×512	∞	0.31	52	1.32
Peppers	512×512	∞	0.34	53	1.37
Average	**--**	∞	**0.32**	**53**	**1.32**

6.7.1.2 Power Consumption

The power consumption of the proposed 2-D DWT architecture is computed using a Xilinx Xpower analyser for selected image sizes and operating frequencies at a supply voltage of 1 Volt, as shown in Figure 6.9. Two frame sizes are considered in this test: 144×176 and 288×352-pixel using a wordlength size of 16-bit (11, 5)-bit. As can be seen in Figure 6.9, the power consumption of the proposed 2-D DWT architecture is from 9 to 48 mW using 20 to 100 MHz operating frequencies. Moreover, the 9 mW dynamic power consumption is small enough to consider the proposed 2-D DWT architecture as a low power consumption architecture.

In the same manner, any other image sizes and/or wordlengths can be used by simply altering the IWL and FWL parameters. However, the power consumption increases when the wordlength and input image sizes are increased. This power increase is due to the extra hardware resources and additional computation operations that will be required.

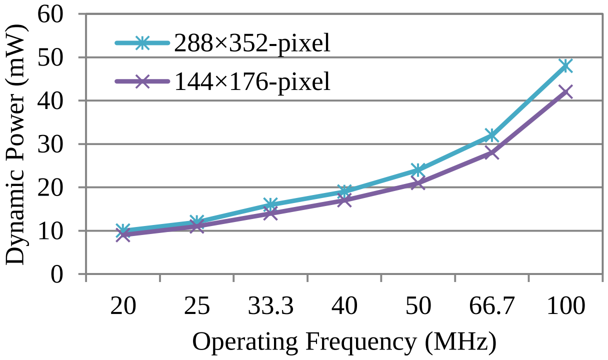


Figure 6.9: Power consumption for various operating frequencies using (11, 5)-bit wordlength.

6.7.1.3 Hardware Usage

The hardware usage, maximum operating frequencies and the computation times of the proposed 2-D DWT architecture are shown in Table 6.3. From this table, a 185 MHz operating frequency can be achieved. This reduces the computation time of the 2-D DWT for an image of 288×352-pixel to 0.55 ms. Furthermore, no block memories have been used in the proposed architecture.

Table 6.3: The hardware usage, maximum operating frequencies and computation time for the proposed 2-D DWT architecture using 144×176 and 288×352-pixel at (11, 5) bit wordlength.

		Available	144×176-pixel	288×352-pixel
Hardware cost	Slice Registers	948,480	6.5K	7.2K
	Slice LUTs	474,240	8.7K	9.5K
	Occupied slices	118,560	2.4K	2.6K
	Bonded IOBs	1,200	73	73
	RAMB36E1/ FIFO36E1s	720	0	0
Maximum frequency (MHz)		--	185	185
Computation time (ms)			0.14	0.55

6.7.2 The 3-D DWT Architecture Performance

6.7.2.1 Rate Distortion Performance

The RMSE, PSNR and the average of the maximum 3-D wavelet coefficients error are computed using 18 and 16-bit wordlengths. The selected (IWL, FWL) pair are set to (11, 7) and (11, 5)-bit. The RMSE and PSNR have been computed between the original and reconstructed frames using (2.12) and (2.13), as shown in Table 6.4 and Table 6.5 for a WL of 18 and 16-bit, respectively. Furthermore, the average of the maximum error between the 3-D wavelet coefficients computed by the proposed architecture and Matlab codes are listed in both tables. Table 6.4 shows that an average PSNR of 79 dB at a wordlength of 18-bit with an average RMSE of 0.03 can be achieved. In such case the average of the maximum error in 3-D wavelet coefficients is less than 0.38. Moreover, in Table 6.5, the average PSNR of the proposed 3-D DWT architecture is 54 dB with an average RMSE of 0.53 using 16-bit wordlength. With this wordlength, the average of the maximum error in the 3-D DWT coefficients is less than 1.49. This minor error is due to the rounding operations required by the proposed shift-add multipliers and other rounding and truncation operations. Moreover, the performance of the proposed architecture can be easily evaluated for any other word size, frame size and number of frames by modifying five input parameters, which are IWL, FWL, P, Q and F.

Table 6.4: The PSNR, RMSE and average of the maximum error for 3-D DWT coefficients using the proposed architectures for WL=18 (11, 7)-bit.

Input sequence	Frame size		PSNR	RMSE	Maximum absolute Error
	P	Q	(dB)		
MRI1	128	128	85	0.015	0.36
MRI2	256	256	77	0.038	0.43
Akiyo	144	176	78	0.033	0.37
Stefan	288	352	79	0.029	0.39
Suzie	144	176	78	0.031	0.35
Bus	288	352	78	0.034	0.39
Flower	288	352	79	0.029	0.41
Calendar	144	176	79	0.030	0.37
Average	-	-	79	0.030	0.38

Table 6.5: The PSNR, RMSE and average of the maximum error for 3-D DWT coefficients using the proposed architectures for WL=16 (11, 5).

Input sequence	Frame size		PSNR	RMSE	Maximum absolute Error
	P	Q	(dB)		
MRI1	128	128	60	0.26	1.41
MRI2	256	256	52	0.61	1.55
Akiyo	144	176	53	0.57	1.49
Stefan	288	352	53	0.56	1.50
Suzie	144	176	53	0.57	1.46
Bus	288	352	53	0.60	1.54
Flower	288	352	53	0.55	1.55
Calendar	144	176	53	0.56	1.40
Average	-	-	54	0.53	1.49

6.7.2.2 Power Consumption Test

The dynamic power consumption of the proposed 3-D DWT architecture is computed using a Xilinx Xpower analyser at a supply voltage of 1 Volt. Two video sequences with frame sizes of 144×167 and 256×256-pixel and various clock frequencies are used, as shown in Figure 6.10. The clock frequency range is selected to be from 20 to 100 MHz and the wordlength is specified as (11, 5)-bit. Figure 6.10 illustrates that the dynamic power consumption of the proposed 3-D DWT architecture ranges from 32 to 159 mW and from 41 to 204 mW for the selected first and second frame sizes, respectively.

As the throughput of the proposed architecture is 4 results/clock cycles, a low operating frequency can be used to obtain the desired frame rate of real time 3-D DWT computation. As an example, in the case of 30 frames per second with a frame size of 144×167-pixel, less than a 1 MHz clock frequency is required as a real time processing speed, and thus, in this case, the dynamic power consumption will drop to 2 mW, as computed using the Xilinx Xpower analyser.

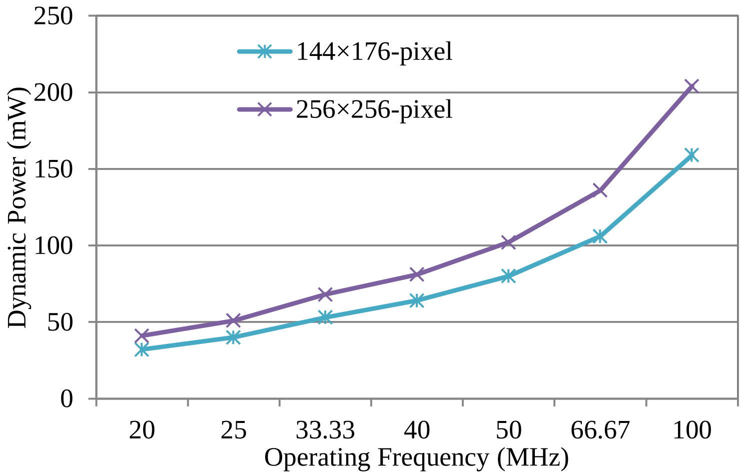


Figure 6.10: Dynamic power consumption (mW) for the proposed 3-D DWT architecture using various operating frequencies using (11, 5)-bit wordlength.

6.7.2.3 Hardware Usage

The hardware usage, operating frequency, throughput and computation times of the proposed 3-D DWT architecture are shown in Table 6.6. Such results are obtained using 144×176 and 256×256-pixel frame sizes, wordlength of (11, 5)-bit and a 10 ns clock period.

Table 6.6: The hardware usage, maximum operating frequencies, throughput and computation time for the proposed 3-D DWT architecture using 144×176 and 256×256 image sizes at (11, 5) bit wordlength.

		Available resources	144×176-pixel	256×256-pixel
Hardware usage	Number of Slice Registers	948,480	31.4K	32.4K
	Number of Slice LUTs	474,240	37.3K	38.8K
	Number of occupied Slices	118,560	10.5K	11K
	Number of bonded IOBs	1,200	97	97
	Number of RAMB36E1/FIFO36E1s	720	0	0
	Number of RAMB18E1/FIFO18E1s	1,440	128	256
Maximum operating frequency (MHz)		--	151.7	139.2
Throughput rate (Pixels/sec×10^6)		--	606.8	556.8
Computation time of eight frames (ms)		--	0.33	0.94

As Table 6.6 demonstrates, 128 and 256-18 Kb block memories have been used in the proposed 3-D DWT architecture for 144×176 and 256×256-pixel frame sizes, respectively. These block memories have been used as a temporary buffer during the computation operation along the frame direction. In addition, a maximum operating frequency of 151 MHz can be achieved with a throughput rate of 606.8 M pixels/second using a 144×176 frame size. Such a high throughput rate has been achieved as a result of using the parallel computation scheme which computes 4 results per clock cycle. Thus, processing times of 0.33 and 0.94 ms are achieved for eight frames in the selected frame sizes, respectively. However, such fast 3-D DWT computation operation comes with extra hardware resources due to the parallel nature of the proposed architecture

6.7.3 Comparison with Other Architectures

The performance of the proposed CDF 9/7 3-D DWT architecture is compared to that of the existing architectures in the literature [27-30, 109] in terms of the hardware usage, computation time and power consumption, as shown in Table 6.7, Table 6.8 and Figure 6.11.

Table 6.7: Comparison with existing 3-D DWT architectures for an input sequence of F-frames each with $P \times P$ pixels *

	Dai [109]	Das [29]	Das [28]	Proposed 3-D DWT architecture
Temporal memory requirements	$2F(K-2)(P+2)+8K$ On chip memory and $FP^2/8$ Off chip memory K: filter length	$P^2/4+4P$ (spatial and temporal)	$5P^2$	$4P^2$
1 level computing time	$\approx FP^2/7$	–	$2P^2+\frac{F}{2}P^2$	$\frac{FP^2}{4}$
1 Level computational latency	–	–	$2P^2$	$P(P+4)+320$
Filter bank	CDF 9/7	Daub-4	CDF 9/7	CDF 9/7
GOF	32 (max)	Infinite	Infinite	Infinite

* Some of the figures in this table are cited from Table I in [28].

As Table 6.7 shows, the proposed CDF 9/7 3-D DWT architecture outperforms the architectures of Dai [109], Das [29] and [28] in terms of memory requirement, computation time and initial latency. From this table, a memory size of four frames $4\mathrm{P}^2$ is required as a temporal buffer in the proposed 3-D DWT architecture, which is used as a frame buffer. This is in contrast to the recent 3-D DWT architecture in [28] which requires a $5\mathrm{P}^2$ temporal buffer. As the size of the input frames is usually more than 144×176 pixels, this memory reduction can be considered an achievement in the proposed architecture.

The number of slices, block RAMs, processing time (C_t) and slice delay-product (SDP) of the proposed 3-D DWT architecture are compared with the architectures in [27, 30, 109], as shown in Table 6.8.

Table 6.8: Comparison between the proposed 3-D DWT and [27, 30, 109] architectures for an input frame size of 144×176-pixel for 6VLX760FF1760-2 Virtex 6 Xilinx FPGA Device

Architecture DWT filter type	Number of frames	Slices	Block RAM	MF (MHz)	C_t (ms)	SDP(s)
Dai [109]	15	10751	64	50.077	1.08	11.61
Based on	30	10467	128	52.578	2.06	21.56
CDF 9/7 filter	60	10607	240	50.121	4.32	45.82
Mohanty [30]	15	28581	48	40.21	0.59	16.86
Based on	30	28581	48	40.21	1.18	33.73
Daub-4 filter	60	28581	48	40.21	2.36	67.45
Mohanty [27]	15	13495	25	88.096	0.539	7.27
Based on	30	13495	25	88.096	1.07	14.44
Daub-4 filter	60	13495	25	88.096	2.15	29.01
Proposed architecture	15	10500	128	151.7	0.80	8.43
Based on	30	10500	128	151.7	1.43	15.04
CDF 9/7 filter	60	10500	128	151.7	2.69	28.26

* The performance of other architectures are cited from Table XI in [27].

The SDP is used to measure the area-time complexity of the architectures as in [27]:

$$SDP = C_t \times Number\ of\ slices \tag{6.41}$$

where C_t is the computation time which is defined in [27]:

$$C_t = Number\ of\ cycles\ required\ for\ the\ computation\ process/MF \tag{6.42}$$

where MF is the maximum operating frequency of the architecture.

In Table 6.8, the proposed 3-D DWT architecture outperforms other architectures in terms of the maximum operating frequency, number of hardware slices and SDP. However, the architecture of [27] shows the lowest memory requirement of all the architectures. Such performance has been achieved for the Daub-4 wavelet filter which is considerably less complex than the CDF 9/7.

The power consumption at 40 MHz clock frequency of the proposed 3-D DWT architecture is compared with those in [27, 30, 109] using a 144×176-pixel frame size and (11,5)-bit wordlength, as shown in Figure 6.11. The data for the remaining architectures are cited from [27]. Hence, Figure 6.11 shows that high power reduction in the proposed 3-D DWT is achieved compared with other architectures. Such an achievement can be assigned to the efficient shift-add multipliers.

In addition, the average of the maximum output error in 3-D wavelet coefficients of the proposed architecture is less than 1.49, while it is not less than 2.7 when the architecture in [27] is run on the Xylophone video sequences [27]. This low error in the proposed architecture can be considered as an additional advantage.

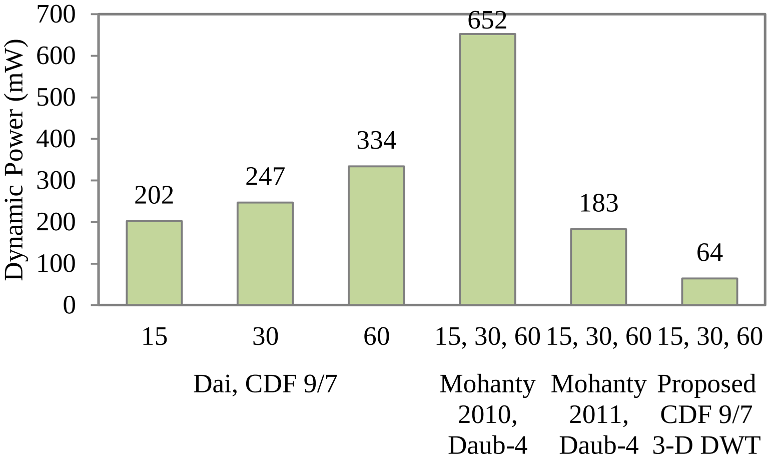


Figure 6.11: Power consumption (mW) of the proposed 3-D DWT and other architectures using 40 MHz clock frequency, (11, 5)-bit wordlength and 144×176 frame size.

6.8 Summary

In this chapter, new multiplierless and parallel lifting-based 1-D, 2-D and 3-D DWT architectures for the CDF 9/7 wavelet filter have been proposed and tested using a 6VLX760FF1760-2 Virtex 6 Xilinx FPGA device. The proposed 1-D, 2-D and 3-D architectures partition the input data into blocks of 4, 4×4 and 4×4×4 pixels, respectively. During each input data block, four temporary values are allocated in a specific registers or memories and four wavelet coefficients are computed. Thus, a high throughput rate and a short processing time have been achieved. Furthermore, shift-add multipliers with negligible error have been suggested to reduce the power consumption arising from using regular multipliers. Full analysis of speed, output accuracy, power consumption and hardware usage for the proposed architectures has been carried. The 2-

D DWT results have revealed that, a processing time of less than 0.55 ms is sufficient to compute the 2-D DWT coefficients for 288×352-pixel. Furthermore, at a wordlength of 18-bit the original data can be recovered from the 2-D DWT coefficients accurately.

In the 3-D DWT architecture, four input frames are simultaneously used, which reduces the temporal buffer to a block memory of four frames only. Such block memory can be constructed using FIFO, single or double port RAMs. The 3-D DWT results have revealed that the proposed 3-D DWT architecture can run at a speed of up to 151 MHz with 4 results/cycle throughput rate. Such a high speed has shortened the 3-D DWT computation time for data size of 144×176×8-pixel to less than 0.33 ms. In addition, at 50 MHz clock frequency, the dynamic power consumption of the proposed 3-D DWT architecture is 80 and 102 mW for 144×176 and 256×256 frame sizes, respectively. Furthermore, the comparisons with other architectures have revealed that the proposed 3-D DWT architecture outperforms similar architectures in the literature in terms of maximum operating frequency, initial latency, power consumption and output accuracy.

Chapter 7: Conclusions and Future Work

7.1 Conclusions

In this thesis, we have addressed the problems of image and video compression algorithms and their related transforms such as power consumption, hardware cost, computation time and output accuracy. Hence, new architectures for image compression algorithms and the related data transforms that considered these issues have been introduced. The original contributions towards the accomplishment of the research objectives as outlined in chapter 2 have been detailed in chapters 3-6, and are summarised below:

1. Two new, low computational complexity non-transform-based algorithms for low bit rate image compression and their architectures have been suggested in chapter 3. The proposed algorithms and architectures are parameterised in terms of the number of quantisation levels, input block size and pipelining stages, offering different output precision levels and processing speeds. The performance evaluation of the new algorithms and their architectures has shown that they are suitable for low power consumption and high speed small devices. The analysis has also revealed that the proposed architectures can operate at a speed of up to 312 MHz. Furthermore, their power consumption is circa 8 mW at an operating frequency of 50 MHz and 4×4-pixel block size.

2. Efficient architectures for multidimensional transforms, such as the DCT and DWT have been suggested in chapters 4-6. The proposed DCT architectures are based on the 1-D Radix-2 DCT and 3-D DCT VR algorithms due to their low computation load. While, the proposed DWT architectures are based on a lifting scheme for CDF 9/7 computation.

3. In chapter 4, two new high speed architectures for multidimensional DCT have been proposed; the first is based on the 1-D DCT Radix-2 algorithm using the RCF computation approach, while the second architecture is based on the 3-D DCT VR algorithm. The results have revealed that a very high computation speed of up to 305 MHz can be achieved. At such high speeds, the 3-D DCT computation times of 512×512×8-point is less than 6.8 ms. In addition; the power consumption using a

wordlength of 21-bit at 10 ns clock period is as low as 98 mW. Furthermore, an infinite PSNR between the original and the reconstructed data using a wordlength of 21-bit can be achieved. The comparisons with similar architectures have revealed that, both outperform existing architectures in terms of power consumption, speed and hardware usage.

4. In chapter 5, two new low hardware usage architectures based on the 3-D DCT VR algorithm have been suggested. The proposed architectures avoid the need for the memory for data transposition in the butterfly and post addition stages, which in turn reduce the hardware usage and improve the processing speed. The proposed architectures are parameterisable in terms of wordlength which provide different output precision levels, power consumption, hardware usage and processing speeds. The proposed architectures have been tested using different images and video sequences, different wordlengths and clock frequencies. The results have revealed that the number of occupied slices is 722 and 1235 for the first and second architecture, respectively. Furthermore, the maximum operating frequencies achieved by the two architectures are 250 and 330 MHz using a 14-bit output wordlength and 8×8×8-pixel input cube size. Furthermore, an infinite PSNR between the original and reconstructed frames using 20-bit wordlength has been attained. Moreover, significant hardware usage reduction with higher operating frequencies is achieved when compared with similar 3-D DCT architectures.

5. New parallel multiplierless lifting-based architectures for 1-D, 2-D and 3-D CDF 9/7 DWT have been suggested, implemented and verified in chapter 6. In such architectures, the constant multipliers have been replaced with their corresponding proposed shift-add multipliers with a negligible error. Also, low memory requirement and high computation speed have been achieved in the proposed architectures.

6. The proposed 2-D CDF 9/7 DWT lifting-based architecture computes the 2-D DWT coefficients by applying 1-D DWT on each row and column using data blocks of 4×4-pixel. In this architecture, all rows in each data block are fed to corresponding 1-D units concurrently. Thus, a high throughput rate and a short processing time have been achieved. The results have revealed that, a computation time of less than 0.55 ms is enough to compute the 2-D DWT coefficients of 288×352-pixel. Furthermore, at a wordlength of 18-bit an exact data can be recovered from the 2-D DWT

coefficients. Furthermore, the power consumption of the proposed architecture using a frame size of 144×167-pixel is as low as 32 mW for a 20 MHz clock frequency.

7. A 3-D DWT parallel architecture has also been proposed in chapter 6 using a separable lifting-based scheme for the CDF 9/7 wavelet filter. In this architecture, four input frames are simultaneously used, which reduces the frame buffer to a block memory of four frames only. The results have shown that the proposed 3-D DWT architecture can run at a speed of up to 151 MHz with 4 results/cycle throughput rate. Such a high speed has reduced the 3-D DWT computation time for data size of 144×176×8-pixel to less than 0.33 ms. Furthermore, the shift-add multiplier replacement had a positive impact on power consumption and has provided a high computation speed. As such, the power consumption of the proposed 3-D DWT architecture is 64 mW at 40 MHz operating frequency and 16-bit wordlength. Moreover, the proposed architectures outperform other similar architectures in terms of hardware usage, speed, throughput rate and latency. Furthermore, the proposed 3-D DWT architecture avoids data computation redundancy compared with similar parallel 3-D DWT architectures in the literature. It is worth mentioning that the output accuracy of all the architectures has been tested and verified using different wordlengths and input data sets. Such evaluation processes revealed that the maximum error in the 3-D DWT coefficient for 16-bit wordlength in the proposed architecture was less than 1.49.

7.2 Future Work

Further work on the following points can be considered:

1. 3-D compression systems based on a 3-D DCT and 3-D DWT architectures can be considered as a possible research direction. Especially for high definition and ultra-high definition video processing and 3-D TV.

2. Another possible research direction is to implement the proposed 3-D DCT and 3-D DWT architectures on the most recent FPGA devices such as family 7 and Zynq device. Full comparison between the performance of the new architectures and this thesis architectures can be performed.

3. A possible research direction is to combine the proposed DCT architectures with a quantisation and encoding algorithm to produce complete DCT-based image and video compression systems. The controllable wordlength and low hardware usage of

our DCT architectures can support such a new research direction. Furthermore, our proposed 3-D DCT architectures can be used to introduce video watermarking, object tracking and medical image compression systems.

4. DWT has been used for MRI images, hyperspectral and video compression algorithms including the EZW, SPIHT, SPECK and EBCOT encoders. Thus, a possible research approach is to associate our high throughput CDF 9/7 DWT architectures with such encoders to introduce a high performance DWT-based compression system.

5. Generalised DWT architectures for other wavelet filters can be introduced using the methodology of our DWT architectures. Such architectures may be targeted to achieve high throughput rates for high order wavelet filters for 2-D and 3-D DWT-based applications.

Appendix A:
Brief Description of Xilinx FPGA Families

Table A.1: Brief description of Xilinx FPGA families introduced since 2003 [125].

Xilinx Family Selected platform	CMOS (nm)	slices per CLB	LUTs per slices	FFs per slices	Maximum Block RAM (kb)	DSP48 slices	DSP slices structure	Block RAM description
Virtex 2 Pro XC2VP100 2003	130	4	2	2	7,992	444	18 × 18 bit multiplier only	18-Kb blocks
Spartan 3 XC3S5000 2003	130	4	2	2	1,872	104		
Virtex 4 XC4VLX200 2005	90	4	2	2	6,048	96	XtremeDSP: 18 × 18 bit multiplier, an adder, and an accumulator	Dual-port 18-Kb RAM blocks
Virtex 5 XC5VSX240T 2006	65	2	4	4	18,576	1,056	DSP48E: 25 × 18 bit multiplier, an adder, and an accumulator	Block RAMs are 36-Kb in size or two independent 18-Kb blocks
Extended Spartan 3A XC3SD3400A 2007	90	4	2	2	2,268	126	DSP48AS : 18 × 18 bit multiplier, an adder, and an accumulator.	18-Kb dual-port blocks
Spartan 6 XC6SLX150T 2009	45	2	4	8	4,824	180	DSP48A1: 18 × 18 bit multiplier, an adder, and an accumulator.	Block RAMs are fundamentally 36-Kb in size. Each block can also be used as two independent 18-Kb blocks.
Virtex 6 XC6VSX475T 2009	40	2	4	8	38,304	2,016	DSP48E1: 25 × 18 bit multiplier, an adder, and an accumulator.	
Artix 7 XC7A200T 2010	28	2	4	8	13,140	740		
Kintex 7 XC7K480T 2010	28	2	4	8	34,380	1,920		
Virtex 7 XC7VX1140T 2010	28	2	4	8	67,680	3,360		

Table A.2: List of Xilinx FPGA families comparison [113].

Features	Artix 7	Kintex 7	Virtex 7	Spartan 6	Virtex 6
Logic Cells	215K	480K	2000K	150K	760K
Block RAM	13Mb	34Mb	68Mb	4.8Mb	38Mb
DSP Slices	740	1,920	3,600	180	2,016
I/O Pins	500	500	1,200	576	1,200
I/O Voltage	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.5V, 1.8V, 2.5V

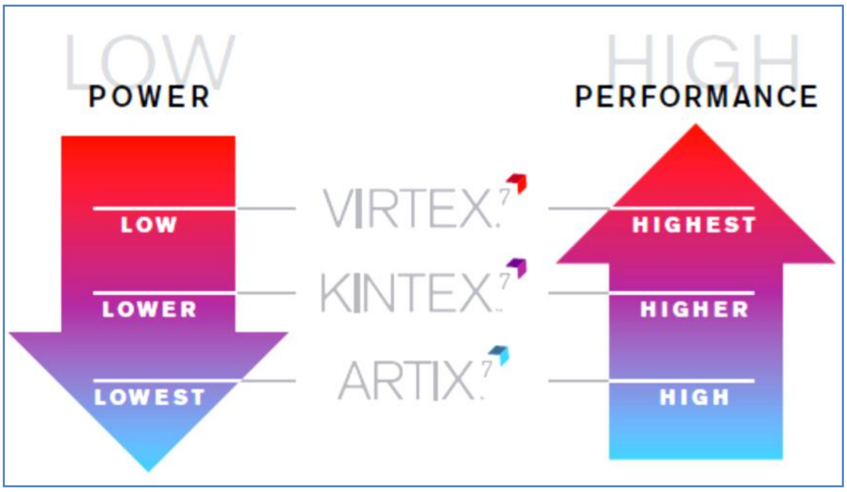


Figure A.1: Series 7 Xilinx FPGA families performance comparison [114].

Appendix B:

Pre-Processing Matlab Codes for IBAQC Model1 and Model2 Architectures

1. The pre-processing Matlab code for Model1 architecture

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% Function  : IBAQC Model1 Architecture
%%%%%% Chapter   : Chapter 3
%%%%%% Programmer: Saad Al-Azawi
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all
clear all
clc
B1=0;
Marg1=0;
[namefile,pathname]=uigetfile({'*.bmp;*.tif;*.tiff;*.jpg;*.jpeg;*.gif'
,'IMAGE Files (*.bmp,*.tif,*.tiff,*.jpg,*.jpeg,*.gif)'},'Chose
GrayScale Image');
[d1,map1]=imread(strcat(pathname,namefile));
[r1,c1,z1]=size(d1);
if z1==3
    d1=rgb2gray(d1);
end

qdc=5; %%% quantization levels
dim=4;
Thresh1=400;
[ri1,ci1,zi1]=size(d1);
if zi1>1
    d1=d1(:,:,2);
end

d1=double(d1);
d1=d1-Marg1;
[r0,c0,z0]=size(d1);
% z3=input('Enter the number of frames = ');
if z0>=8
    z3=8;
else
    z3=1;
end
d2=d1(:,:,1:z3);

[r1,c1,z1]=size(d2);
z2=z1;
d3=d2(:,:,1:z2);
[rx,cx,zx]=size(d3);

dim3=z2;
k1=0;
dimall=dim*dim*dim3;
idx=1;
for k=1:dim3:z2
    k1=k+dim3-1;
    j1=0;
    i1=0;
    for i=1:dim:r1
        i1=i+dim-1;
        for j=1:dim:c1
            j1=j+dim-1;
            X=d3(i:i1,j:j1,k:k1);
            X1=reshape(X,dimall,1);
            idx1=idx+dimall-1;
            InData(idx:idx1,2)=X1;
            idx=idx1+1;
        end
    end
end

[r3,c3,z3]=size(InData);
InData(:,1)=0:r3-1; %%%%%% Architecture Input data

save InData InData;
[r3,c3,z3]=size(InData);

I2=InData;
I1=I2;
[rt1,ct1]=size(I1);
N=dim;
k1=0;
k2=0;
for k=1:N*N:rt1-1
    k2=k2+1;
    k1=k+N*N-1;
    Iz=I1(k:k1,2);
    varz(k2,1)=(mean(Iz.^2)-(mean(Iz))^2);

    varz(k2,2)=max(Iz);
    varz(k2,3)=min(Iz);
    varz(k2,4)=max(Iz)-min(Iz);
    varz(k2,5)=std(Iz);
end
varz;
stdz=sqrt(varz(:,1));
[r1,c1]=size(I1);
%%%%%%%%%%%%%%%%%%%%%%%%%% Division modification
levqdc=2^qdc;
Xs=[0:255]/levqdc;
Xs=ceil(Xs);
divx1=Xs';
Xs=1./Xs;
Xs(1,1)=0;
divx=Xs';
Bpoint=8; %%% Binary point
BAcc1=8+log2(N^2);
BShft1=8+Bpoint; %% 5  for binary points  13/5
BMult1=16+Bpoint;
BAcc2=16+log2(N^2);
BShft2=16+Bpoint; %% 5  for binary points
% %%%  Sim Time
sim_tim=2*dim^2+ri1*ci1+11-1;
```

2. The pre-processing Matlab code for Model2 architecture

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% Function  : IBAQC Model2 Architecture
%%%%%% Chapter   : Chapter 3
%%%%%% Programmer: Saad Al-Azawi
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clear all
clc
B1=0;
Marg1=0;
[namefile,pathname]=uigetfile({'*.bmp;*.tif;*.tiff;*.jpg;*.jpeg;*.gif'
,'IMAGE Files (*.bmp,*.tif,*.tiff,*.jpg,*.jpeg,*.gif)'},'Chose
GrayScale Image');
[d1,map1]=imread(strcat(pathname,namefile));
[r1,c1,z1]=size(d1);
if z1==3
    d1=rgb2gray(d1);
end
qdc=5; %%% quantization levels
dim=8;
Thresh1=2;
[ri1,ci1,zi1]=size(d1);
if zi1>1
    d1=d1(:,:,2);
end
d1=double(d1);
d1=d1-Marg1;
[r0,c0,z0]=size(d1);
% z3=input('Enter the number of frames = ');
if z0>=8
    z3=8;
else
    z3=1;
end
d2=d1(:,:,1:z3);

[r1,c1,z1]=size(d2);
% z2=fix(z1/8)*8;
z2=z1;
d3=d2(:,:,1:z2);
[rx,cx,zx]=size(d3);
dim3=z2;
k1=0;
dimall=dim*dim*dim3;
idx=1;
for k=1:dim3:z2
    k1=k+dim3-1;
    j1=0;
    i1=0;
    for i=1:dim:r1
        i1=i+dim-1;
        for j=1:dim:c1
            j1=j+dim-1;
            X=d3(i:i1,j:j1,k:k1);
            X1=reshape(X,dimall,1);
            idx1=idx+dimall-1;
            InData(idx:idx1,2)=X1;
            idx=idx1+1;
        end
    end
```

```
end

[r3,c3,z3]=size(InData);
InData(:,1)=0:r3-1;
save InData InData;
[r3,c3,z3]=size(InData);
I2=InData;
I1=I2;
[rt1,ct1]=size(I1);
N=dim;
k1=0;
k2=0;
for k=1:N*N:rt1-1
    k2=k2+1;
    k1=k+N*N-1;
    Iz=I1(k:k1,2);
    varz(k2,1)=(mean(Iz.^2)-(mean(Iz))^2);
    varz(k2,2)=max(Iz);
    varz(k2,3)=min(Iz);
    varz(k2,4)=max(Iz)-min(Iz);
    varz(k2,5)=std(Iz);
end
varz;
stdz=sqrt(varz(:,1));
[r1,c1]=size(I1);
%%%%%%%%%%%%%%%%%%%%%%%%% program for divider
levqdc=2^qdc;
Xs=[0:255]/levqdc;
divx1=Xs';
Xs=ceil(Xs);
divx2=Xs';
Xs=1./Xs;
Xs(1,1)=0;
divx=Xs';
Bpoint=8; %%% Binary point
BAcc1=8+log2(N^2);
BShft1=8+Bpoint; %% 5  for binary points  13/5
sim_tim=2*dim^2+ri1*ci1+7-1;
```

Appendix C:

Memory Writing/Reading addresses of the proposed 3-D DCT VR Architecture

This appendix presents the memory writing/reading addresses at each point of the proposed 3-D DCT VR architecture. The actual addresses at each node in Figure 4.12 and Figure 4.17 (P_1, P_2... P_{13}) are computed by adding the content of ROMs M_2(N-words) to each element in M_1(N^2-word). The elements in each table are stored in M_2 row by row from the element 0 till the last element in the bottom right of each table.

The following example illustrates the procedure of addresses computation for the input data to the first butterfly stage (Node P_1 in Figure 4.12), which are computed from Table C.1 as shown below:

- The addresses of the first eight samples of input data: the content of M_2 (0, 16, 64, 80, 256, 272, 320 and 336) is added to the first element in M_1 (element 0), then the addresses are: 0, 16, 64, 80, 256, 272, 320 and 336.
- The addresses of the second eight input data: the content of M_2 is added to the second element in M_1 (element 1), so the addresses are: 1, 17, 65, 81, 257, 273, 321 and 337.
- The addresses of the third eight input data: the content of M_2 is added to the third element in M_1 (element 8), so the addresses are: 8, 24, 72, 88, 264, 280, 328, 344 and so on until the last element in M_1 (element 175).

In the rest of this appendix, the actual content of M_1 and M_2 to generate the reading/writing addresses at each node of the 3-D DCT VR architecture is explained for N=8. The Memory Writing Controller is labelled as (MWC) and it is constructed as shown in Figure 4.16 using different β values, as shown underneath each MWC block in Figure 4.12 and Figure 4.17. The memory reading addresses are generated using a combination of MRE and address generator units. The MRE units are shown in Figure 4.11, while the address generator unit is shown in the shaded part Figure 4.10. The constant value (α variable in Figure 4.11 of MRE unit) required to commence the memory reading operation is shown underneath each Memory Reading Controller (MRC) block in Figure 4.12 and Figure 4.17.

1. Addresses at Node P_1

The addresses at node P_1 are generated from the reordering stage (TMem1 reading addresses generator unit) using the data shown in Table C.1.

Table C.1: Content of M_1 and M_2 for memory reading addresses for TMem1 (node P_1).

M_2 Content	M_1 Content							
0	0	1	8	9	2	3	10	11
16	32	33	40	41	34	35	42	43
64	4	5	12	13	6	7	14	15
80	36	37	44	45	38	39	46	47
256	128	129	136	137	130	131	138	139
272	160	161	168	169	162	163	170	171
320	132	133	140	141	134	135	142	143
336	164	165	172	173	166	167	174	175

2. Addresses at Node P_2 and P_3

The addresses at P_2 (writing addresses of TMem_2) is the same as that at P_1 but is delayed by *(β+1)* clock cycles. TMem2 reading addresses (addresses at node P_3) are generated from Table C.2 using the same procedure discussed earlier.

Table C.2: Content of M_1 and M_2 for memory reading addresses for TMem2 (node P_3).

M_2 Content	M_1 Content							
0	0	1	2	3	4	5	6	7
8	16	17	18	19	20	21	22	23
32	64	65	66	67	68	69	70	71
40	80	81	82	83	84	85	86	87
128	256	257	258	259	260	261	262	263
136	272	273	274	275	276	277	278	279
160	320	321	322	323	324	325	326	327
168	336	337	338	339	340	341	342	343

3. Addresses at Node P_4 and P_5

The writing addresses of TMem3 (P_4) are the same as those at P_3 but are delayed by *(β+1)* clock cycles, while the memory reading addresses of TMem3 (P_5) are $0, 1, 2, 3 \ldots N^3 - 1$. These are generated using an MRE-II unit and a single 10-bit counter (address generator).

4. Addresses at Node P_6 and P_7

The data is written to TMem4 in bit reverse order (BRO) using a BRO address controller, as shown in Figure 4.12. The BRO address controller block is composed of a 10-bit counter, two ROMs (M_1 and M_2) and three slice elements (address generator unit) and MWC with *β*=36.The BRO addresses (TMem4 writing addresses) at P_6 are computed from Table C.3.

Table C.3: Content of M_1 and M_2 for memory reading addresses for TMem4 (node P_6).

M_2 Content	M_1 Content							
0	0	128	64	192	16	144	80	208
256	8	136	72	200	24	152	88	216
32	2	130	66	194	18	146	82	210
288	10	138	74	202	26	154	90	218
4	1	129	65	193	17	145	81	209
260	9	137	73	201	25	153	89	217
36	3	131	67	195	19	147	83	211
292	11	139	75	203	27	155	91	219

The memory reading address generator unit of TMem4 (MRC4 at node P_7) is composed of a 10-bit counter, three slice elements and a concatenation block, which can be represented by the following relation:

$$S_3 = X_n \ for\ 1 \leq n \leq 6,\ S_2 = X_n\, for\ \ 7 \leq n \leq 9\,, S_1 = X_n\, for\, n = 10$$

$$\text{then}\ \ R\_adds = [S_1\, S_3\, S_2] \qquad \text{(C.1)}$$

where X_n is the binary representation of the 10 bits input counter, $n = 1,2,\ldots log_2 2N^3$, where $n = 1$ represents the LSB, and R_adds is the required memory reading addresses for the input data to the first post addition stage.

5. Addresses at Node P$_8$ and P$_9$

The P$_7$ addresses are used at the TMem5 writing phase (P$_8$) with *(β+1)* clock cycle delay while the memory reading addresses at P$_9$ are generated using a 10-bit counter from 0 $to\ N^3 - 1$ as an Adds_generator unit.

6. Addresses at Node P$_{10}$ and P$_{11}$

The TMem6 writing addresses at P$_{10}$ are the P$_9$ addresses delayed by *(β+1)* clock cycles, and its reading addresses (P$_{11}$) are generated using a 10 bits counter, four slice elements and a concatenation block, as described by the following relation:

$$S_4 = X_n\ for\ 1 \leq n \leq 3\ \ , S_3 = X_n\ for\ 4 \leq n \leq 6\ , S_2 = X_n\ for\ 7 \leq n \leq 9\ ,$$

$$S_1 = X_n\ for\ n = 10$$

$$\text{then}\ \ R\_dds = [S_1\ \ S_4\ S_3\ S_2] \qquad \text{(C.2)}$$

7. Addresses at Node P$_{12}$ and P$_{13}$

A delayed version of P$_{11}$ is used in the TMem7 writing addresses at P$_{12}$, whereas the TMem7 reading addresses of the normalisation and finalization process at node P$_{13}$ are $0, 1, 2, \ldots., N^3 - 1$, which are generated using a 10-bit counter as an address generator unit together with MRE-II unit.

Appendix D:

Pre-Processing Matlab Codes for Chapter 4; Model1 and Model2 Architectures

1. The pre-processing Matlab code for Model1 architecture

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% Function  : Model1 Architecture
%%%%%% Chapter   : Chapter 4
%%%%%% Programmer: Saad Al-Azawi
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all
clear all
clc
% B1=0 3 5;
 B1=5;
VidFil=3;
Siz1=8;
switch VidFil
    case 1
        load mri
    case 2
        load mristack
        D=mristack;
    case 3
        load AkiyoRgb;
        D=AkiyoRgb(:,:,2,1:Siz1);
    case 4
        load Stefan1;
        D=Stefan1(:,:,2,1:Siz1);
    case 5
        load Suzie;
        D=suzie148(:,:,2,1:Siz1);

    case 6
        load Bus;
        D=BusColor(:,:,2,1:Siz1);
    case 7
        load Flower;
        D=FlowerCIF(:,:,2,1:Siz1);
        clear FlowerCIF;

    case 8
        load mobile;
        D=MobileQ300(:,:,2,1:Siz1);
end

d1=squeeze(D);
d1=double(d1);
[r0,c0,z0]=size(d1);
z3=8;
d2=d1(:,:,1:z3);
d2(:,:,1)=d1(:,:,2);
d2(:,:,2)=d1(:,:,1);

d1=d2;
```

```
[r1,c1,z1]=size(d2);
z2=fix(z1/8)*8;
d3=d2(:,:,1:z2);
dim=8;
k1=0;
dimall=dim*dim*dim;
idx=1;
for k=1:dim:z2
    k1=k+7;
    j1=0;
    i1=0;
    for i=1:dim:r1
        i1=i+7;
        for j=1:dim:c1
            j1=j+7;
            X=d3(i:i1,j:j1,k:k1);
            X1=reshape(X,dimall,1);
            idx1=idx+511;
            InData(idx:idx1,2)=X1;
            idx=idx1+1;
        end
    end
end

[r3,c3,z3]=size(InData);
InData(:,1)=0:r3-1;
save InData InData;

I2=InData;
I1=I2;

[r1,c1]=size(I1);
InData=I1;
save InData InData;
M=8;
%%%%%%%%%%%%%%%%%     dS1  delay   %   FIRST       %%%%%%%%%%%%%stage
dS1(1,1)=5;
shd1=1;
dS1(2,1)=5;
shd2=1;
dS1=dS1-2;
dS1(3,1)=3;
shd3=1;
dS1(4,1)=3;
dS1(5,1)=0;
MuxD1=1;  %%multiplexer Delay Stage 1
M_W_st1=(sum(dS1))*M+MuxD1+8;  %%% start of Memory writing _ Stage One
R1D=50; %% dealy between Write and read operation of the first stage
%%%%%%%%%%%%%%%%%     dS2  delay %%%%%%%%%%%%%%%%%%%%%      SECOND
dS2(1,1)=5;
s2hd1=1;  %%%% shift in the combination of add+multiply ...
dS2(2,1)=5;
s2hd2=1;  %%%% shift in the combination of add+multiply ...
dS2=dS2-2;
dS2(3,1)=3;
s2hd3=1;  %%%% shift in the combination of add+multiply ...
dS2(4,1)=3;
dS2(5,1)=0;
 MuxD2=1;  %%multiplexer Delay Stage 2
M_W_st2=(sum(dS2))*M+MuxD2+8;

%%%%%%%%%%%%%%%%%     dS3  delay %%%%%     Third       %%%%stage
dS3(1,1)=5;
```

```
s3hd1=1;  %%%% shift in the combination of add+multiply ...
dS3(2,1)=5;
s3hd2=1;  %%%% shift in the combination of add+multiply ...
dS3=dS3-2;
dS3(3,1)=3;
s3hd3=1;  %%%% shift in the combination of add+multiply ...

dS3(4,1)=3;
dS3(5,1)=0;

MuxD3=1;  %%multiplexer Delay Stage 3
R2D=10;
% M_W_st3=M_W_st2+(sum(dS3))*M+23+378+15;  %%% start of Memory writing
operation_ Stage Three
M_W_st3=(sum(dS3))*M+MuxD2+8;  %%% start of Memory writing operation_
Stage Three
sim_tim=M_W_st1+M_W_st2+M_W_st3+956+r1-1 %%% simulation time
NLen=32; %% element length for all stage except stage 1
Nsub=13;%%% mantesa for all stage except stage 1
NLen1=10; %%% First stage             // first column of addres only
Nsub1=0;  %%% Mentesa First stage  // first column of addres only
NLen=21-B1; %% element length for all stage except stage 1
Nsub=9-B1;%%% mantesa for all stage except stage 1
```

2. The pre-processing Matlab code for Model2 architecture

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% Function   : Model2 Architecture
%%%%%% Chapter    : Chapter 4
%%%%%% Programmer: Saad Al-Azawi
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clear all
clc
B1=3;
VidFil=3;
Siz1=8;
switch VidFil
    case 1
        load mri
    case 2
        load mristack
        D=mristack;
    case 3
        load AkiyoRgb;
        D=AkiyoRgb(:,:,2,1:Siz1);
    case 4
        load Stefan1;
        D=Stefan1(:,:,2,1:Siz1);
    case 5
        load Suzie;
        D=suzie148(:,:,2,1:Siz1);

    case 6
        load Bus;
        D=BusColor(:,:,2,1:Siz1);
    case 7
        load Flower;
        D=FlowerCIF(:,:,2,1:Siz1);
        clear FlowerCIF;
    case 8
        load mobile;
        D=MobileQ300(:,:,2,1:Siz1);
end

D(D>255)=255;
d1=squeeze(D);
d1=double(d1);
[r0,c0,z0]=size(d1);
% z3=input('Enter the number of frames = ');
z3=8;
d2=d1(:,:,1:z3);
[r1,c1,z1]=size(d2);
z2=fix(z1/8)*8;
d3=d2(:,:,1:z2);
dim=8;
k1=0;
dimall=dim*dim*dim;
idx=1;
for k=1:dim:z2
    k1=k+7;
    j1=0;
    i1=0;
    for i=1:dim:r1
        i1=i+7;
        for j=1:dim:c1
            j1=j+7;
            X=d3(i:i1,j:j1,k:k1);
            X1=reshape(X,dimall,1);
            idx1=idx+511;
            InData(idx:idx1,2)=X1;
            idx=idx1+1;
        end
    end
end
%%%%%%%%%%%%%%%%%%% converting to 255 !!!!!!
O=InData(:,2);
[r3,c3,z3]=size(InData);
InData(:,1)=0:r3-1;
save InData InData;
Iin=InData;
 [r1,c1]=size(Iin);
Sim_time=2860+r1-1;  %For last model Foward_Final-7
load Iads
load stg0ads8 ; %64 adss strting for writing address
load stg0offset; % 8 offsets for writing address
load Initadds;  % this is the whole 512 adss  ( NOT USED)
load offs2
load st8stg1
%xlswrite('st8stg1.xlsx',st8stg1);
load stg1ads8  %%% For FPGA one Column = st8stg1(:,4)-1;
%%%%%%%%%%%%%%%% STAGE 2
load st8stg2;
load stg2ads8  %%% For FPGA one Column = st8stg2(:,4)-1;
load offstg2;
load st8stg3;
load stg3ads8  %%% For FPGA one Column = st8stg3(:,4)-1;
load offstg3;

%%%% STAGE 4 bIT REVERSE ORDER
load StBRO1; %% the start for writing the output of stage 3 addresses
for each 8 element (the memory content will be in Reverse Order.
load offBRO1; % the offset
load Zads  % offset  of Z (third)  adder

load NormF2;  %the last stage normalization factors

%%%% new idea for twiddle factors

load TWFAC1
load TWFAC2
load TWFAC3;
NLen=32; %% element length for all stage except stage 1
Nsub=13;%%% mantesa for all stage except stage 1
NLen1=10; %%% First stage
Nsub1=0;  %%% Mentesa First stage
NLen=32-B1; %% element length for all stage except stage 1
Nsub=13-B1;%%% mantesa for all stage except stage 1
NLen=21-B1; %% element length for all stage except stage 1
Nsub=9-B1;%%% mantesa for all stage except stage 1
%%%% Choose the Model
%%% Model for a and c   Mod1=1, for b    Mod1=2
    dS0=0;
    dS0(1,1)=1;
    dS0(2,1)=1;
    dS0(3,1)=1;
    delS0=sum(dS0)*8; %%%% the delay required by first stage

    dS1=0;
    dS1(1,1)=1;
    dS1(2,1)=1;
    dS1(3,1)=1;
    delS1=sum(dS1)*8;  %%%% the delay required by Second stage
    dS2=0;
    dS2(1,1)=1;
    dS2(2,1)=1;
    dS2(3,1)=1;
    delS2=sum(dS2)*8;  %%%% the delay required by Third stage
    %%%% Butterfly
    dS3=0;
    dS3(1,1)=1;
    dS3(2,1)=1;
    dS3(3,1)=1;
    delS3=sum(dS3)*8;  %%%% the delay required by first stage
    dS4=0;
    dS4(1,1)=1;
    dS4(2,1)=1;
    dS4(3,1)=1;
    delS4=sum(dS4)*8;  %%%% the delay required by Second stage
    dS5=0;
    dS5(1,1)=1;
    dS5(2,1)=1;
    dS5(3,1)=1;
    delS5=sum(dS5)*8;  %%%% the delay required by Third stage
    MuxD1=1;

totdel=delS0+delS1+delS2+delS3+delS4+delS5;
totdel1=2844+totdel;
Sim_time=totdel1+r1-1;
```

Appendix E:

Pre-Processing Matlab Codes for Chapter 5; Model1 and Model2 Architectures

1. The pre-processing Matlab code for Model1 architecture

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% Function  : Model1 Architecture
%%%%%% Chapter   : Chapter 5
%%%%%% Programmer: Saad Al-Azawi
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clear all
clc
VidFil=8;
Siz1=8;
switch VidFil
    case 1
        load mri
    case 2
        load mristack
        D=mristack;
    case 3
        load AkiyoRgb;
        D=AkiyoRgb(:,:,2,1:Siz1);
    case 4
        load Stefan1;
        D=Stefan1(:,:,2,1:Siz1);
    case 5
        load Suzie;
        D=suzie148(:,:,2,1:Siz1);

    case 6
        load Bus;
        D=BusColor(:,:,2,1:Siz1);
    case 7
        load Flower;
        D=FlowerCIF(:,:,2,1:Siz1);
        clear FlowerCIF;

    case 8
        load mobile;
        D=MobileQ300(:,:,2,1:Siz1);
end
B1=0;
d1=squeeze(D);
d1=double(d1);
[r0,c0,z0]=size(d1);
% z3=input('Enter the number of frames = ');
z3=8;
d1=d1(:,:,1:z3);
d2=d1(:,:,1:z3);
[r1,c1,z1]=size(d2);
z2=fix(z1/8)*8;

d3=d2(:,:,1:z2);
```

```
dim=8;
% in0=double(d1(75:75+dim-1,73:73+dim-1,1:dim));
k1=0;
dimall=dim*dim*dim;
idx=1;
for k=1:dim:z2
    k1=k+7;
    j1=0;
    i1=0;
    for i=1:dim:r1
        i1=i+7;
        for j=1:dim:c1
            j1=j+7;
            X=d3(i:i1,j:j1,k:k1);
            X1=reshape(X,dimall,1);
            idx1=idx+511;
            InData(idx:idx1,2)=X1;
            idx=idx1+1;
        end
    end
end
O=InData(:,2);
[r3,c3,z3]=size(InData);
InData(:,1)=0:r3-1;
save InData InData;
Iin=InData;
[r1,c1]=size(Iin);
Sim_time=2860+r1-1;  %For last model Foward_Final-7

%%%  Twiddle factors of stage 1
TFstg1=[1;
    1;
    1;
    1;
    1.961570561;
    1.111140466;
    -0.390180644;
    -1.662939225;
    ];
TFstg1=TFstg1/2;
%%%  Twiddle factors of stage 2
TFstg2=[1
    1;
    1.847759065;
    -0.765366865;
    ];
TFstg2=TFstg2/2;
%%%  Twiddle factors of stage 3
TFstg3=[1;
1.414213562;
2;
2.828427125;
    ];
TFstg3=TFstg3/2;%*64;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
NLen0=10; %% integer
NSub0=0;%%% Mantissa
NLen0=NLen0+NSub0;
%%%% Stage 2
NLen1=11; %%
NSub1=2;%%%
NLen1=NLen1+NSub1;
%%%% Stage 3
```

```
NLen2=NLen1+1; %%
NSub2=2;%%%

LS01=1;
LS02=1;
LS03=1;
LS0=LS01+LS02+LS03;
LMult0=2;

LS11=1;
LS12=1;
LS13=1;
LS1=LS11+LS12+LS13;
LMult1=6;

LS21=1;
LS22=1;
LS23=1;
LS2=LS21+LS22+LS23;
LMult2=4;

load BROWadds;
load BRORadds;
load ReordWadds;
load ReordRadds;
load ReordRaddsA;
% Second stage -twiddle factors
load Twiddle2;
% Twiddle2=Twiddle2/8;
load TwiddleA;
Sim_tim=1135+r3;
```

2. The pre-processing Matlab code for Model2 architecture

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% Function   : Model2 Architecture
%%%%%% Chapter    : Chapter 5
%%%%%% Programmer: Saad Al-Azawi
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clear all
clc

VidFil=8;
Siz1=8;
switch VidFil
    case 1
        load mri
    case 2
        load mristack
        D=mristack;
    case 3
        load AkiyoRgb;
        D=AkiyoRgb(:,:,2,1:Siz1);
    case 4
        load Stefan1;
        D=Stefan1(:,:,2,1:Siz1);
    case 5
        load Suzie;
        D=suzie148(:,:,2,1:Siz1);
    case 6
        load Bus;
        D=BusColor(:,:,2,1:Siz1);
    case 7
        load Flower;
        D=FlowerCIF(:,:,2,1:Siz1);
        clear FlowerCIF;

    case 8
        load mobile;  %calendar
        D=MobileQ300(:,:,2,1:Siz1);
end
%%%%%%%%%%%%%%%%%%%%%%Input Data
B1=0;
d1=squeeze(D);
d1=double(d1);
[r0,c0,z0]=size(d1);
% z3=input('Enter the number of frames = ');
z3=8;
d1=d1(:,:,1:z3);
d2=d1(:,:,1:z3);
[r1,c1,z1]=size(d2);
z2=fix(z1/8)*8;
d3=d2(:,:,1:z2);
dim=8;
% in0=double(d1(75:75+dim-1,73:73+dim-1,1:dim));
k1=0;
dimall=dim*dim*dim;
idx=1;
for k=1:dim:z2
    k1=k+7;
    j1=0;
    i1=0;
    for i=1:dim:r1
        i1=i+7;
        for j=1:dim:c1
            j1=j+7;
            X=d3(i:i1,j:j1,k:k1);
            X1=reshape(X,dimall,1);
            idx1=idx+511;
            InData(idx:idx1,2)=X1;
            idx=idx1+1;
        end
    end
end

[r3,c3,z3]=size(InData);
InData(:,1)=0:r3-1;
save InData InData;
Iin=InData;
[r1,c1]=size(Iin);
%%%  Twiddle factors of stage 1
TFstg1=[1;
    1;
    1;
    1;
    1.961570561;
    1.111140466;
    -0.390180644;
    -1.662939225;
    ];
```

```
TFstg1=TFstg1/2;
%%%  Twiddle factors of stage 2
TFstg2=[1
    1;
    1.847759065;
    -0.765366865;
    ];
TFstg2B=TFstg2(3:4,1);  %%%Model 2 BTFLstage 2 lower part
TFstg2=TFstg2/2;
load TFstg2A;  %%%Model 2 BTFL stage 2 upper part
%%%%%%%%%^^^^ in the architecture this will multiplied three times i.e
the factor will  be 8
%%%  Twiddle factors of stage 3
TFstg3=[1;
    1.414213562;
    2;
    2.828427125;
    ];
TFstg3=TFstg3/2;%*64;
TFstg3A=TFstg3(1:3,1);
TFstg3B=TFstg3(2:4,1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
NLen0=10; %% integer
NSub0=0;%%% Mantissa
NLen0=NLen0+NSub0;
%%%% Stage 2
NLen1=11; %%
NSub1=8;%%%
NLen1=NLen1+NSub1;
%%%% Stage 3
NLen2=NLen1+1; %%
NSub2=8;%%%
LS01=1;
LS02=1;
LS03=1;
LS0=LS01+LS02+LS03;
LMult0=2;
LS11=1;
LS12=1;
LS13=1;
LS1=LS11+LS12+LS13;
LMult1=6;
LS21=1;
LS22=1;
LS23=1;
LS2=LS21+LS22+LS23;
LMult2=4;
%%%%%%%%%%%%%%%%%%%%%%%%%%-
load BROWadds;
load BRORadds;
load ReordWadds;
load ReordRadds;
load ReordRaddsA;

% Second stage 64 twiddle factors
load Twiddle2;
load TwiddleA;
Sim_tim=954+r3+1;
```

Appendix F:

The 2-D and 3-D DWT Architectures Controllers

This Appendix presents the data controllers for the proposed 2-D and 3-D DWT architectures.

1. The Proposed 2-D DWT Architecture controller

The main controller of the 2-D DWT architecture generates the following controlling signals, Row Selector (SelR) and Column Selector (SelC), as shown in Figure F.1 and Figure F.2. The SelR and SelC signals are used to set/reset the selectors inside all RUs and CUs. Hence, considering an input image with P × Q pixels, the Counter1 in Figure F.1, is a $\left\lceil log_2\left(\frac{Q}{4}\right)\right\rceil$-bit counter. This counter controls the transition at the boundaries of each row in each 4×4-pixel block. Counter2 is used to control the transitions at the column boundaries between blocks. It is a $\left\lceil log_2\left(\frac{P}{4}\times\frac{Q}{4}\right)\right\rceil$-bit. Counter1 and Comparator1 produce the SelR signal which can be represented as follows:

$$SelR = \begin{cases} 0 & where\ j = 0 \\ 1 & otherwise \end{cases} \qquad \text{(F.1)}$$

where $j = 0,1,2, \ldots \frac{Q}{4} - 1$.

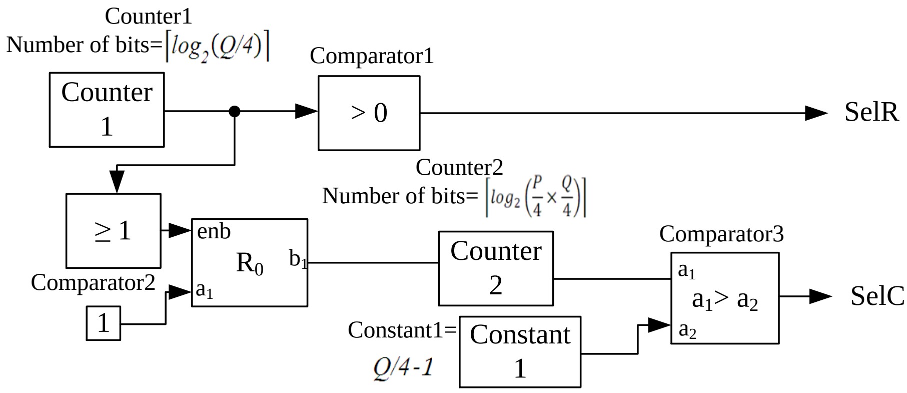
Accordingly, Counter2 together with Comparator3 and Constant1, Constant1= $\frac{Q}{4} - 1$, are used to generate the SelC signal to drive the selectors of all CUs. Such signal can be represented as follows:

$$SelC = \begin{cases} 0 & where\ i = 0 \\ 1 & otherwise \end{cases} \qquad \text{(F.2)}$$

where $i = 0,1,2, \ldots \frac{P}{4} - 1$.

The controlling signals for an image tile of 8×8 are shown in Figure F.2. Such waveforms represent a down sampled version by 4 from the original signals for clarity.

In summary, a quick look into Figure F.1 shows that the main feature of the proposed controller is its simplicity and ability to tackle any image size.



$P \times Q$=Frame size (Rows×Columns) and ⌈… …⌉ =Ceiling function.

R_0= Register

Figure F.1: The 2-D DWT controller.

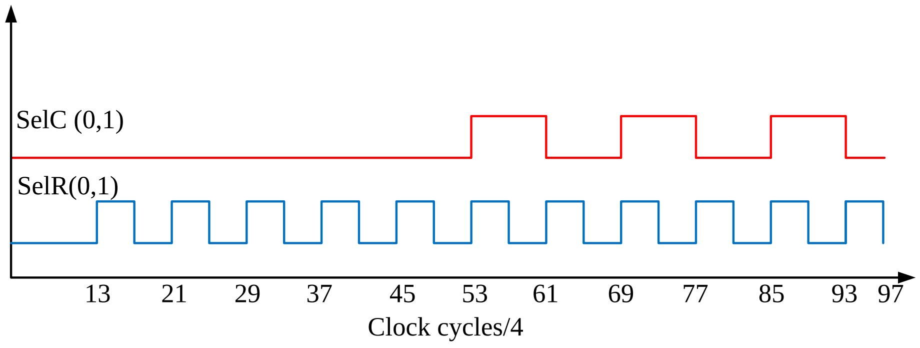
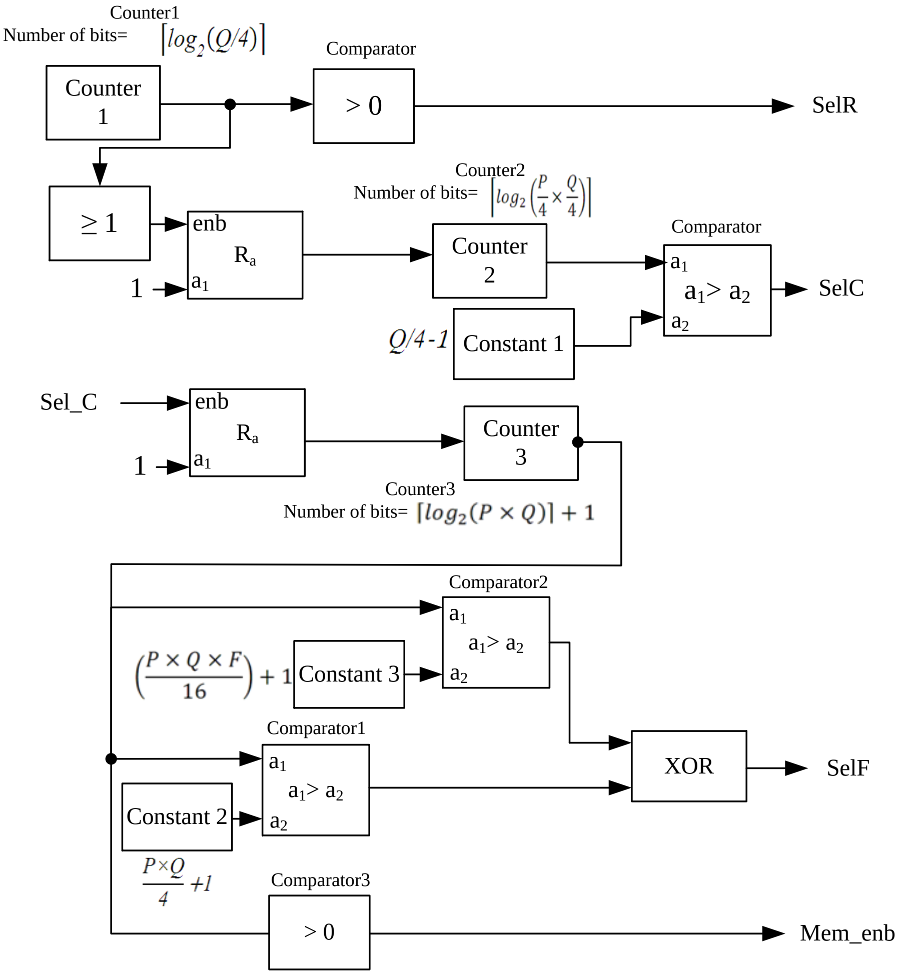


Figure F.2: Down sampled 2-D DWT controller signals.

2. The Proposed 3-D DWT Architecture Controller

The 3-D DWT controller is constructed similar to the 2-D DWT controller, with the only difference being an additional circuit which is used to control frame direction computation process, as shown in Figure F.3.

The proposed 3-D DWT controller consists of a Row-Column controller, comparators and 2-bit counter (counter4). The controller outputs are SelR and SelC for Row-Column DWT (2-D DWT) and SelF and Mem_Enb to control the frame direction computation process. The first two signals are explained in the 2-D DWT controller circuit, and the SelF and Mem_Enb signals are responsible for frame direction controlling and enabling functions. The SelF signal is used to drive the selectors that control the transitions at the beginning and end of the whole 3-D input stream (*Ctrl* blocks in each FU). The SelF signal is turned on during the period that starts at the beginning of the second GOF (second computation round; $= 1$). It is used to activate the reading operation of the temporary values that allocated in the frame buffer during the first GOF computation round ($n = 0$). Two comparators and an XOR logic gate are used for this purpose; the first comparator (Comparator1 in Figure F.3) generates an enabling signal after $Q(P + 4) + 293$ clock cycles to drive on the selectors in each frame unit. Then, the second comparator output (Comparator2 in Figure F.3) is turned on after $Q\left(\frac{PF}{4} + 4\right) + 293$ clock cycles. Therefore, the XOR output which represents the SelF signal is turned on from $Q(P + 4) + 293$ to $Q\left(\frac{PF}{4} + 4\right) + 293$ clock cycles. The computation period of the boundary GOFs, the first and last GOF, is accomplished when the SelF signal is turned to zero; SelF=0. Thus, at the last GOF, it becomes zero for the last $QP + 27$ clock cycles from the computation time. The main selector waveforms down sampled by 4 are shown in Figure F.4, for clarity.



$P \times Q$=Frame size (Rows×Columns), F= Number of frames and $\lceil \ldots \ldots \rceil$ =Ceiling function

R$_e$: register

Figure F.3: Controller block diagram of the proposed 3-D DWT architecture.

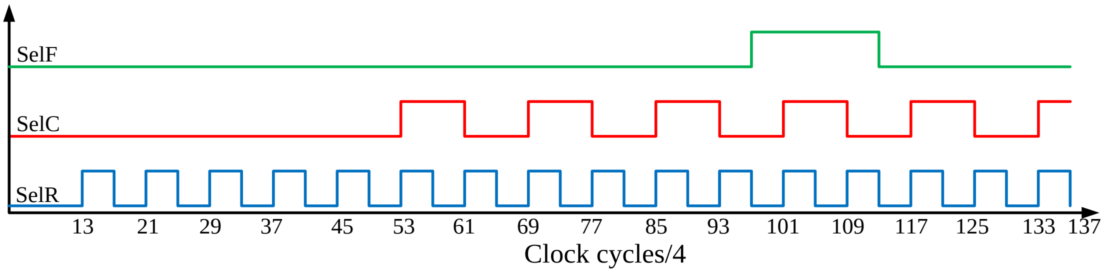


Figure F.4: The main signals of the 3-D DWT controller down sampled by 4.

The second output signal from the controller is the Mem-Enb signal. It is used to enable/disable the memory bank and its addressing unit.

In summary, the main characteristics of the proposed controller are its low complexity and ability to cope with different frame sizes and numbers with an infinite GOF. This controller arrangement supports the power consumption reduction by turning on and off memories and other elements at a required time.

Appendix G:

Pre-Processing Matlab Codes for Chapter 6; 2-D DWT and 3-D DWT Architectures

1. The pre-processing Matlab code for 2-D DWT architecture

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% Function  : 2-D DWT Architecture
%%%%%% Chapter   : Chapter 6
%%%%%% Programmer: Saad Al-Azawi
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clear all
clc
VidFil=3;
Siz1=8;
switch VidFil
    case 1
        load mri
    case 2
        load mristack
        D=mristack;
    case 3
        load AkiyoRgb;
        D=AkiyoRgb(:,:,2,1:Siz1);
    case 4
        load Stefan1;
        D=Stefan1(:,:,2,1:Siz1);
    case 5
        load Suzie;
        D=suzie148(:,:,2,1:Siz1);
    case 6
        load Bus;
        D=BusColor(:,:,2,1:Siz1);
    case 7
        load Flower;
        D=FlowerCIF(:,:,2,1:Siz1);
        clear FlowerCIF;
    case 8
        load mobile;
        D=MobileQ300(:,:,2,1:Siz1);
    case 9

[namefile,pathname]=uigetfile({'*.png;*.bmp;*.tif;*.tiff;*.jpg;*.jpeg;
*.gif','IMAGE Files
(*.png,*.bmp,*.tif,*.tiff,*.jpg,*.jpeg,*.gif)'},'Chose GrayScale
Image');
        [d1,map1]=imread(strcat(pathname,namefile));
        [rx1,cx1,zx1]=size(d1);
        if zx1==3
            D=d1(:,:,2);
        else
            D=d1(:,:,1);
        end
end
d1=abs(double(squeeze(D)));
d1=zeros(512,512);
```

```
d1=d1(:,:,1,1);
save d1 d1;
z3=1;
d1=d1(:,:,1);
d1=double(d1);
[rx,cx,zx]=size(d1);
idx=1;
for k=1:4:zx
    j1=0;
    i1=0;
    for i=1:4:rx
        i1=i+3;
        for j=1:4:cx
            j1=j+3;
            idx1=idx+16-1;
            X=d1(i:i1,j:j1,1);
            X1=reshape(X',16,1);
            InData(idx:idx1,1)=X1;
            idx=idx1+1;
        end
    end
end
Seq=0:idx1-1;
Seq=Seq';
IF1=[Seq InData(:,1)];
clear InData
RowS=rx;  %% number of Rows
ColS=cx;  %% number of Columns
RowC=ceil(log2(ColS/4));  %%% number of bits for Rows counter
ColC=ceil(log2(RowS/4*ColS/4)); %%% number of bits for Columns counter
DelS=ColS/4;  %%% the required delay of folded data in the columns
unit= number of blocks in each row
PeriodC=ColS/4-1;  %%% used to comupte the constant required to
control the columns operation
INShift=0;%%% Normalization Input Shift
RBShift=0;  %%% Normalization Row Shift
% Row Stage
BpointR=7; %% Binary points
B1LenR=11+BpointR;
% Column Stage
BpointC=7; %% Binary points
B1LenC=11+BpointC;
Bpoint=7; %% Binary points
B1Len=11+Bpoint;
BpointR=5; %% Binary points
B1LenR=11+BpointR;
% Column Stage
BpointC=5; %% Binary points
B1LenC=11+BpointC;
BpointC=5; %% Binary points
B1LenC=11+BpointC;
Bpoint=5; %% Binary points
B1Len=11+Bpoint;
AF=-0.630463621;
BF=0.743750255;
CF=-0.668067178;
DF=0.638443853;
K0F=2.421021123;
K1F=2.065244222;
sim_tim=289+ColS*4+RowS*ColS-2
```

2. The pre-processing Matlab code for 3-D DWT architecture

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% Function   : 3-D DWT Architecture
%%%%%% Chapter    : Chapter 6
%%%%%% Programmer: Saad Al-Azawi
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clear all
clc
VidFil=1;
Siz1=8;
switch VidFil
    case 1
        load mri
    case 2
        load mristack
        D=mristack;
    case 3
        load AkiyoRgb;
        D=AkiyoRgb(:,:,2,1:Siz1);
    case 4
        load Stefan1;
        D=Stefan1(:,:,2,1:Siz1);
    case 5
        load Suzie;
        D=suzie148(:,:,2,1:Siz1);

    case 6
        load Bus;
        D=BusColor(:,:,2,1:Siz1);
    case 7
        load Flower;
        D=FlowerCIF(:,:,2,1:Siz1);
        clear FlowerCIF;

    case 8
        load mobile;
        D=MobileQ300(:,:,2,1:Siz1);
end
d1=double(d1(:,:,1:Siz1));
save d1 d1
[rx,cx,zx]=size(d1);
idx=1;
for k=1:4:zx
    j1=0;
    i1=0;
    for i=1:4:rx
        i1=i+3;
        for j=1:4:cx
            j1=j+3;
            idx1=idx+16-1;
            for k1=1:4
                X=d1(i:i1,j:j1,k1+k-1);
                X1=reshape(X',16,1);
                InData(idx:idx1,k1)=X1;
            end
            idx=idx1+1;
        end
    end
end
Seq=0:idx1-1;
```

```
Seq=Seq';
IF1=[Seq InData(:,1)];
IF2=[Seq InData(:,2)];
IF3=[Seq InData(:,3)];
IF4=[Seq InData(:,4)];
RowS=rx;  %% number of Rows
ColS=cx;  %% number of Columns
FraS=zx;  %% number of Frames
RowC=ceil(log2(ColS/4));  %%% number of bits for Rows counter
ColC=ceil(log2(RowS/4*ColS/4)); %%% number of bits for Columns counter
FraC=ceil(log2(RowS/4*ColS/4*FraS/4)); %%number of bits Frame counter
DelS=ColS/4;  %%% the required delay of folded data in the columns
unit= number of blocks in each row
DelF=RowS*ColS/4; %%% the required delay of folded data in the columns
unit= number of blocks in each Frame
PeriodC=ColS/4-1;  %%% used to comupte the constant required to
control the columns operation
PeriodF=RowS/4*ColS/4-1;
AF=-0.630463621;
BF=0.743750255;
CF=-0.668067178;
DF=0.638443853;
K0F=2.421021123;
K1F=2.065244222;
Bpoint=5; %% Binary points  %%%% This is used for all models bfore 7
B1Len=11+Bpoint;
% Row Stage
BpointR=5; %% Binary points
B1LenR=11+BpointR;
% Column Stage
BpointC=5; %% Binary points
B1LenC=11+BpointC;
Tim1=289+ColS*4+RowS*ColS-2+39; % 39 is the delay after the 2-D
sim_tim delay when finish the first 2-D frame
Tim2=Tim1+idx1;
sim_tim=Tim2-2
MuxTim=rx/2*cx/2*(zx/4-1)-1;
```

References

[1] J. M. Guo and C. C. Su, "Improved block truncation coding using extreme mean value scaling and block-based high speed direct binary search," *IEEE Signal Processing Letters,* vol. 18, pp. 694-697, 2011.

[2] E. Delp and O. Mitchell, "Image compression using block truncation coding," *IEEE Transactions on Communications,* vol. 27, pp. 1335-1342, 1979.

[3] J. M. Guo and Y. F. Liu, "High capacity data hiding for error-diffused block truncation coding," *IEEE Transactions on Image Processing,* vol. 21, pp. 4808-4818, 2012.

[4] J. Wang, Y. C. Jeung, and J. W. Chong, "Improved block truncation coding using low cost approach for color image compression," in *ACM International Conference Proceeding Series*, 2009, pp. 1106-1109.

[5] J. Wang, K. Min, and J. Chong, "A hybrid image coding in overdriving for motion blur reduction in LCD," in *Lecture Notes in Computer Science* vol. 4740, ed, 2007, pp. 263-270.

[6] G. K. Wallace, "The JPEG still picture compression standard," *IEEE Transactions on Consumer Electronics,* vol. 38, pp. xviii-xxxiv, 1992.

[7] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG 2000 still image compression standard," *IEEE Signal Processing Magazine,* vol. 18, pp. 36-58, 2001.

[8] A. Madisetti and A. N. Willson, Jr., "DCT/IDCT processor design for HDTV applications," in *International Symposium on Signals, Systems, and Electronics (ISSSE '95)*, 1995, pp. 63-66.

[9] J. Mathews, M. S. Nair, and L. Jo, "Improved BTC algorithm for gray scale images using K-means quad clustering," vol. 7666 LNCS, ed, 2012, pp. 9-17.

[10] R. E. Atani, M. Baboli, S. Mirzakuchaki, S. E. Atani, and B. Zamanlooy, "Design and implementation of a 118 MHz 2D DCT processor," in *IEEE International Symposium on Industrial Electronics*, 2008, pp. 1076-1081.

[11] M. Jridi and A. Alfalou, "A low-power, high-speed DCT architecture for image compression: Principle and implementation," in *18th IEEE/IFIP VLSI System on Chip Conference (VLSI-SoC)*, 2010, pp. 304-309.

[12] M. El Aakif, S. Belkouch, N. Chabini, and M. M. Hassani, "Low power and fast DCT architecture using multiplier-less method," in *2011 Faible Tension Faible Consommation (FTFC)*, 2011, pp. 63-66.

[13] B. Z. Guo, L. Niu, and Z. M. Liu, "Implementation of 2-D DCT based on FPGA," in *Proceedings of SPIE - The International Society for Optical Engineering*, 2010.

[14] H. L. P. A. Madanayake, R. J. Cintra, D. Onen, V. S. Dimitrov, and L. T. Bruton, "Algebraic integer based 8×8 2-D DCT architecture for digital video processing," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2011, pp. 1247-1250.

[15] A. M. Shams, A. Chidanandan, W. Pan, and M. A. Bayoumi, "NEDA: A low-power high-performance DCT architecture," *IEEE Transactions on Signal Processing*, vol. 54, pp. 955-964, 2006.

[16] G. K. a. S. V. Khurram Bukhari, "DCT and IDCT implementations on different FPGA technologies," *Computer Engineering Lab, Delft University of Technology*, 2009

[17] E. D. Kusuma and T. S. Widodo, "FPGA implementation of pipelined 2D-DCT and quantization architecture for JPEG image compression," in *2010 International Symposium in Information Technology (ITSim)*, 2010, pp. 1-6.

[18] W. Zhang, Z. Jiang, Z. Gao, and Y. Liu, "An efficient VLSI architecture for lifting-based discrete wavelet transform," *IEEE Transactions on Circuits and Systems II: Express Briefs,* vol. 59, pp. 158-162, 2012.

[19] X. Tian, L. Wu, Y. H. Tan, and J. W. Tian, "Efficient multi-input/multi-output vlsi architecture for two-dimensional lifting-based discrete wavelet transform," *IEEE Transactions on Computers,* vol. 60, pp. 1207-1211, 2011.

[20] C. Cheng and K. K. Parhi, "High-speed VLSI implementation of 2-D discrete wavelet transform," *IEEE Transactions on Signal Processing,* vol. 56, pp. 393-403, 2008.

[21] C. Zhang, C. Wang, and M. O. Ahmad, "A pipeline VLSI architecture for fast computation of the 2-D discrete wavelet transform," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 59, pp. 1775-1785, 2012.

[22] B. K. Mohanty and P. K. Meher, "Memory efficient modular VLSI architecture for highthroughput and low-latency implementation of multilevel lifting 2-D DWT," *IEEE Transactions on Signal Processing,* vol. 59, pp. 2072-2084, 2011.

[23] B. K. Mohanty, A. Mahajan, and P. K. Meher, "Area- and power-efficient architecture for high-throughput implementation of lifting 2-D DWT," *IEEE Transactions on Circuits and Systems II: Express Briefs,* vol. 59, pp. 434-438, 2012.

[24] S. Saponara, "Real-time and low-power processing of 3D direct/inverse discrete cosine transform for low-complexity video codec," *Journal of Real-Time Image Processing,* pp. 1-11, 2012.

[25] Y. Ikegaki, T. Miyazaki, and S. G. Sedukhin, "3D-DCT processor and its FPGA implementation," *IEICE Transactions on Information and Systems,* vol. E94-D, pp. 1409-1418, 2011.

[26] A. Ahmad and A. Amira, "Efficient reconfigurable architectures for 3D medical image compression," in *International Conference on Field-Programmable Technology*, 2009, pp. 472-474.

[27] B. K. Mohanty and P. K. Meher, "Memory-efficient architecture for 3-D DWT using overlapped grouping of frames," *IEEE Transactions on Signal Processing,* vol. 59, pp. 5605-5616, 2011.

[28] A. Das, A. Hazra, and S. Banerjee, "An efficient architecture for 3-D discrete wavelet transform," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 20, pp. 286-296, 2010.

[29] B. Das and S. Banerjee, "Data-folded architecture for running 3-D DWT using 4-tap Daubechies filters," *IEE Proceedings: Circuits, Devices and Systems,* vol. 152, pp. 17-24, 2005.

[30] B. K. Mohanty and P. K. Meher, "Parallel and pipeline architectures for high-throughput computation of multilevel 3-D DWT," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 20, pp. 1200-1209, 2010.

[31] C. C. Wang, C. H. Chen, and I. H. Chen, "Modified VQ-BTC algorithm for image compression," *Electronics Letters,* vol. 34, pp. 1390-1392, 1998.

[32] R. Ngadiran, "Rate scalable image compression in the wavelet domain," *PhD thesis, Newcastle university,* 2012.

[33] A. Ouafi, A. Taleb Ahmed, Z. Baarir, and A. Zitouni, "A modified Embedded Zerotree Wavelet (MEZW) algorithm for image compression," *Journal of Mathematical Imaging and Vision,* vol. 30, pp. 298-307, 2008.

[34] A. Ouafi, A. T. Ahmed, Z. Baarir, N. Doghmane, and A. Zitouni, "Color image coding by modified Embedded Zerotree Wavelet (EZW) algorithm," in *2nd Information and Communication Technologies (ICTTA '06),* 2006, pp. 1451-1456.

[35] S. Al-Azawi, S. Boussakta, and A. Yakovlev, "Image compression algorithms using intensity based adaptive quantization coding," *American Journal of Engineering and Applied Sciences,* vol. 4, pp. 504-512, 2011.

[36] L. N. Faria, L. M. G. Fonseca, and M. H. M. Costa, "Performance evaluation of data compression systems applied to satellite imagery," *Journal of Electrical and Computer Engineering,* // 2012.

[37] E. Oshri, N. Shelly, and H. B. Mitchell, "Interpolative three-level block truncation coding algorithm," *Electronics Letters,* vol. 29, pp. 1267-1268, 1993.

[38] S. Domnic, "BTC image compression with variable length integer codes," in *IET International Conference on Visual Information Engineering (VIE),* 2006, pp. 184-188.

[39] S. Al-Azawi, S. Boussakta, and A. Yakovlev, "Low complexity image compression algorithm using AMBTC and bit plane squeezing," in *7th International Workshop on Systems, Signal Processing and their Applications (WOSSPA),* 2011, pp. 131-134.

[40] J. M. Guo, "Improved block truncation coding using modified error diffusion," *Electronics Letters,* vol. 44, pp. 462-464, 2008.

[41] J. M. Guo and M. F. Wu, "Improved block truncation coding based on the void-and-cluster dithering approach," *IEEE Transactions on Image Processing,* vol. 18, pp. 211-213, 2009.

[42] J. Wang, K. Y. Min, Y. C. Jeung, and J. W. Chong, "Improved BTC using luminance bitmap for color image compression," in *Proceedings of the 2009 2nd International Congress on Image and Signal Processing (CISP'09),* 2009.

[43] J. Wang and J. W. Chong, "High performance overdrive using improved motion adaptive codec in LCD," *IEEE Transactions on Consumer Electronics,* vol. 55, pp. 20-26, 2009.

[44] J. W. Han, Y. J. Yoon, T. S. Wang, M. C. Hwang, and S. J. Ko, "Improved block truncation coding for color image compression in LCD overdrive," in *Proceedings of the International Symposium on Consumer Electronics (ISCE),* 2008.

[45] J. Wang and J. W. Chong, "Adaptive multi-level block truncation coding for frame memory reduction in LCD overdrive," *IEEE Transactions on Consumer Electronics,* vol. 56, pp. 1130-1136, 2010.

[46] J. Park and S. Lee, "Structural similarity based image compression for LCD overdrive," *IEEE Transactions on Consumer Electronics,* vol. 58, pp. 1276-1284, 2012.

[47] Y. C. Chou and H. H. Chang, "A data hiding scheme for color image using BTC compression technique," in *9th IEEE International Conference on Cognitive Informatics (ICCI),* 2010, pp. 845-850.

[48] W. Hong, J. Chen, T. S. Chen, and C. W. Shiu, "Steganography for block truncation coding compressed images using hybrid embedding scheme," *International Journal of Innovative Computing, Information and Control,* vol. 7, pp. 733-743, 2011.

[49] W. Hong, T. S. Chen, C. W. Shiu, and M. C. Wu, "Lossless data embedding in BTC codes based on prediction and histogram shifting," vol. 65, ed: Applied Mechanics and Materials, 2011, pp. 182-185.

[50] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE Transactions on Computers,* vol. C-23, pp. 90-93, 1974.

[51] D. J. Le Gall, "The MPEG video compression standard," in *Compcon Spring '91: Digest of Papers,* 1991, pp. 334-335.

[52] W. Li, "Overview of fine granularity scalability in MPEG-4 video standard," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 11, pp. 301-317, 2001.

[53] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 13, pp. 560-576, 2003.

[54] S. Boussakta and H. O. Alshibami, "Fast algorithm for the 3-D DCT-II," *IEEE Transactions on Signal Processing,* vol. 52, pp. 992-1001, 2004.

[55] S. C. Chan and K. L. Ho, "Direct methods for computing discrete sinusoidal transforms," *IEE Proceedings on Radar and Signal Processing,* vol. 137, pp. 433-442, 1990.

[56] O. Alshibami and S. Boussakta, "Fast algorithm for the 3D DCT," in *IEEE International Conference on Acoustics, Speech, and Signal Processing Proceedings (ICASSP '01),* 2001, pp. 1945-1948 vol.3.

[57] M. C. Lee, R. K. W. Chan, and D. A. Adjeroh, "Fast three-dimensional discrete cosine transform," *SIAM Journal on Scientific Computing,* vol. 30, pp. 3087-3107, 2008.

[58] D. A. Adjeroh and S. D. Sawant, "Error-resilient transmission for 3D DCT coded video," *IEEE Transactions on Broadcasting,* vol. 55, pp. 178-189, 2009.

[59] S. Sawant and D. A. Adjeroh, "Balanced multiple description coding for 3D DCT video," *IEEE Transactions on Broadcasting,* vol. 57, pp. 765-776, 2011.

[60] T. Fryza, "A complete video coding chain based on multi-dimensional discrete cosine transform," *Radioengineering,* vol. 19, pp. 421-428, 2010.

[61] M. Bhaskaranand and J. D. Gibson, "Distributions of 3D DCT coefficients for video," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* 2009, pp. 793-796.

[62] H. Tang, W. Sun, B. Gao, and F. Zhang, "Research on quantization and scanning order for 3-D DCT video coding," in *International Conference on Computer Science and Electronics Engineering (ICCSEE)*, 2012, pp. 200-204.

[63] R. Westwater and B. Furht, "The XYZ algorithm for real-time compression of full-motion video," *Real-Time Imaging*, vol. 2, pp. 19-34, 1996.

[64] M. A. Engin and B. Cavusoglu, "New approach in image compression: 3D spiral JPEG," *IEEE Communications Letters*, vol. 15, pp. 1234-1236, 2011.

[65] Y. Zhang and H. Bi, "Transparent video watermarking exploiting spatio-temporal masking in 3D-DCT domain," *Journal of Computational Information Systems*, vol. 7, pp. 1706-1713, 2011.

[66] J. Yang, J. Hu, and P. Liu, "Video watermarking by 3D DCT," in *7th International Conference on Signal Processing Proceedings (ICSP '04)*, 2004, pp. 861-864.

[67] F. Yong-Gang, "Robust video watermarking scheme based on 3D DCT," in *International Conference onComputer Application and System Modeling (ICCASM)*, 2010, pp. V11-635-V11-638.

[68] A. Karami, S. Beheshti, and M. Yazdi, "Hyperspectral image compression using 3D discrete cosine transform and support vector machine learning," in *11th International Conference on Information Science, Signal Processing and their Applications (ISSPA)*, 2012, pp. 809-812.

[69] X. Li, A. Dick, C. Shen, A. van den Hengel, and H. Wang, "Incremental learning of 3D-DCT compact representations for robust visual tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PP, pp. 1-1, 2012.

[70] L. Jin, A. Boev, A. Gotchev, and K. Egiazarian, "3D-DCT based perceptual quality assessment of stereo video," in *18th IEEE International Conference on Image Processing (ICIP)*, 2011, pp. 2521-2524.

[71] M. Joachimiak, D. Rusanovskyy, M. M. Hannuksela, and M. Gabbouj, "Multiview 3D video denoising in sliding 3D DCT domain," in *Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*, 2012, pp. 1109-1113.

[72] M. Kim Yong and Z. Li Hong, "A lip reading method based on 3-D DCT and 3-D HMM," in *2011 International Conference on Electronics and Optoelectronics (ICEOE)*, 2011, pp. V1-115-V1-119.

[73] S. G. Mallat, "Theory for multiresolution signal decomposition: the wavelet representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 674-693, 1989.

[74] B. E. Usevitch, "A tutorial on modern lossy wavelet image compression: foundations of JPEG 2000," *IEEE Signal Processing Magazine*, vol. 18, pp. 22-35, 2001.

[75] I. Daubechies, "Ten lectures on wavelets," *CBMS, SIAM, 61*, 1995.

[76] K. Sayood, "Introduction to data compression," *Third Eddition, Elsevier Inc.*, 2006.

[77] I. Adam, "Complex Wavelet Transform:application to denoising," *PhD thesis, Politehnica University of Timisoara and Université de Rennes 1*, 2012.

[78] A. Shahbahrami, "Algorithms and architectures for 2D discrete wavelet transform," *Journal of Supercomputing,* vol. 62, pp. 1045-1064, 2012.

[79] A. S. Remya Ajai and N. Nagaraj, "A novel methodology for memory reduction in distributed arithmetic based discrete wavelet transform," in *Procedia Engineering*, 2012, pp. 226-233.

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

[81] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *Journal of Fourier Analysis and Applications,* vol. 4, pp. x1-268, 1998.

[82] A. N. Skodras, C. A. Christopoulos, and T. Ebrahimi, "JPEG2000: The upcoming still image compression standard," *Pattern Recognition Letters,* vol. 22, pp. 1337-1345, 2001.

[83] A. Aggoun, "Compression of 3D integral images using 3D wavelet transform," *Journal of Display Technology,* vol. 7, pp. 586-592, 2011.

[84] N. Sriraam and R. Shyamsunder, "3-D medical image compression using 3-D wavelet coders," *Digital Signal Processing,* vol. 21, pp. 100-109, 2011.

[85] B. M. Sunil and C. P. Raj, "Analysis of wavelet for 3D-DWT volumetric image compression," in *Proceedings - 3rd International Conference on Emerging Trends in Engineering and Technology (ICETET),* 2010, pp. 180-185.

[86] A. Ahmad, B. Krill, A. Amira, and H. Rabah, "3D Haar wavelet transform with dynamic partial reconfiguration for 3D medical image compression," in *IEEE Biomedical Circuits and Systems Conference (BioCAS 2009),* 2009, pp. 137-140.

[87] P. Bao and X. Ma, "Image adaptive watermarking using wavelet dOmain singular value decomposition," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 15, pp. 96-102, 2005.

[88] Y. Wan, Y. Yang, and Q. Q. Chen, "Multitone block truncation coding," *Electronics Letters,* vol. 46, pp. 414-415, 2010.

[89] M. Ayinala and K. K. Parhi, "Parallel pipelined FFT architectures with reduced number of delays," in *Proceedings of the ACM Great Lakes Symposium on VLSI (GLSVLSI),* 2012, pp. 63-66.

[90] O. Nibouche, S. Boussakta, M. Darnell, and M. Benaissa, "Algorithms and pipeline architectures for 2-D FFT and FFT-like transforms," *Digital Signal Processing: A Review Journal,* vol. 20, pp. 1072-1086, 2010.

[91] M. Ayinala, M. Brown, and K. K. Parhi, "Pipelined parallel FFT architectures via folding transformation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 20, pp. 1068-1081, 2012.

[92] O. Nibouche, S. Boussakta, and M. Darnell, "Pipeline Architectures for Radix-2 New Mersenne Number Transform," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 56, pp. 1668-1680, 2009.

[93] S. An and C. Wang, "Recursive algorithm, architectures and FPGA implementation of the two-dimensional discrete cosine transform," *IET, Image Processing* vol. 2, pp. 286-294, 2008.

[94] G. Jiun-In and L. Chih-Chen, "A generalized architecture for the one-dimensional discrete cosine and sine transforms," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 11, pp. 874-881, 2001.

[95] Z. Wu, J. Sha, Z. Wang, L. Li, and M. Gao, "An improved scaled DCT architecture," *IEEE Transactions on Consumer Electronics,* vol. 55, pp. 685-689, 2009.

[96] C. Yuan-Ho and C. Tsin-Yuan, "A high performance video transform engine by using space-time scheduling strategy," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* vol. 20, pp. 655-664, 2012.

[97] A. Aggoun and I. Jalloh, "Two-dimensional DCT/IDCT architecture," *IEE Proceedings on Computers and Digital Techniques,* vol. 150, pp. 2-10, 2003.

[98] J. I. Guo, "Efficient parallel adder based design for one-dimensional discrete cosine transform," *IEE Proceedings on Circuits, Devices and Systems,* vol. 147, pp. 276-282, 2000.

[99] S. Al-Azawi, S. Boussakta, and A. Yakovlev, "High precision and low power DCT architectures for image compression applications," in *IET Conference on Image Processing (IPR)*, 2012, pp. 1-6.

[100] I. Jalloh, A. Aggoun, and M. McCormick, "3D DCT architecture for compression of integral 3D images," in *IEEE Workshop on Signal Processing Systems, SiPS: Design and Implementation*, 2000, pp. 238-244.

[101] A. Aggoun and I. Jalloh, "A parallel 3D DCT architecture for the compression of integral 3D images," in *The 8th IEEE International Conference on Electronics, Circuits and Systems (ICECS)* 2001, pp. 229-232 vol.1.

[102] M. Bakr and A. E. Salama, "Implementation of 3D-DCT based video Encoder/Decoder system," in *Midwest Symposium on Circuits and Systems*, 2002, pp. II13-II16.

[103] S. Saponara, L. Fanucci, and P. Terreni, "Low-power VLSI architectures for 3D discrete cosine transform (DCT)," in *IEEE 46th Midwest Symposium on Circuits and Systems*, 2003, pp. 1567-1570 Vol. 3.

[104] D. U. Lee, L. W. Kim, and J. D. Villasenor, "Precision-aware self-quantizing hardware architectures for the discrete wavelet transform," *IEEE Transactions on Image Processing*, vol. 21, pp. 768-777, 2012.

[105] A. Ahmad, B. Krill, A. Amira, and H. Rabah, "Efficient architectures for 3D HWT using dynamic partial reconfiguration," *Journal of Systems Architecture,* vol. 56, pp. 305-316, 2010.

[106] X. Xiaodong and Z. Yiqi, "Efficient FPGA implementation of 2-D DWT for 9/7 float wavelet filter," in *International Conference on Information Engineering and Computer Science (ICIECS),* 2009, pp. 1-4.

[107] H. Yu, L. Qing, M. Siwei, and C. C. J. Kuo, "A VLSI architecture for a fast computation of the 2-D Discrete Wavelet Transform," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2007, pp. 3980-3983.

[108] S. Barua, J. E. Carletta, K. A. Kotteri, and A. E. Bell, "An efficient architecture for lifting-based two-dimensional discrete wavelet transforms," *The VLSI Journal, Integration,* vol. 38, pp. 341-352, 2005.

[109] Q. Dai, X. Chen, and C. Lin, "A novel VLSI architecture for multidimensional discrete wavelet transform," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, pp. 1105-1110, 2004.

[110] N. H. Ja’afar, A. Ahmad, and A. Amira, "Rapid prototyping of three-dimensional transform for medical image compression," in *11th International Conference on Information Science, Signal Processing and their Applications (ISSPA)*, 2012, pp. 842-847.

[111] C. Maxfield, "The design warior's guide to FPGAs," vol. ISBN 0-7506-7604-3, 2004.

[112] H. Sarmento, "Virtex platforms comparison," *Technical Report, 2008* [Online]. Available: http://prosys.inesc-id.pt/UWBR/Reports/UWBR-virtex.pdf, Accessed on 24/11/2012.

[113] Xilinx, "Xilinx 7 series " *Xilinx FPGA Families, 2012,* [Online]. Available: http://www.xilinx.com/products/silicon-devices/fpga/index.htm, Accessed on 25/11/2012.

[114] Xilinx, "Xilinx 7 series FPGAs-Brief " *XILINX , 2012,* [Online]. Available:http://www.xilinx.com/publications/prod_mktg/7-Series-Product-Brief.pdf, Accessed on 18/12/2012.

[115] Xilinx, "Zynq-7000 all programmable SoC overview," *Advance Product Specification, 2012,* [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf, Accessed on 25/11/2012.

[116] Xilinx, "XA Zynq-7000 all programmable SoC overview," *Advance Product Specification, 2012,* [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds188-XA-Zynq-7000-Overview.pdf, Accessed on 25/11/2012.

[117] Xilinx, "Zynq-7000 family use cases and markets," *Xilinx Web site,* [Online]. Available: http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/use-cases-and-markets/index.htm, Accessed on 26/11/2012.

[118] Xilinx, "Virtex-5 family overview," *Product Specification, 2009,* [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf, Accessed on 21/11/2012.

[119] Xilinx, "Virtex-6 family overview," *Product Specification, 2012,* [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf, Accessed on 21/11/2012.

[120] Xilinx, "Virtex-5 FPGA user guide," *User Guide, 2012,* [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug190.pdf, Accessed on 22/11/2012.

[121] Xilinx, "Virtex-6 FPGA configurable logic block user guide," *User Guide, 2012,* [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug364.pdf, Accessed on 22/11/2012.

[122] Xilinx, "Virtex-6 FPGA memory resources," *User Guide, 2011,* [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug363.pdf, Accessed on 22/11/2012.

[123] Xilinx, "Virtex-6 FPGA DSP48E1 slice," *User Guide, 2011,* [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug369.pdf, Accessed on 22/11/2012.

[124] Xilinx, "Xilinx CORE Generator System," *Xilinx, 2013,* [Online]. Available: http://www.xilinx.com/tools/coregen.htm, Accessed on 10/06/2013.

[125] Xilinx, "Xilinx documentation," *Documentation, 2012*, [Online]. Available: http://www.xilinx.com/support/#nav=sd-nav-link-182711&tab=tab-sd, Accessed on 20/11/2012.

[126] Xilinx, "What is a FPGA?," [Online]. Available: http://www.origin.xilinx.com/fpga/, Accessed on 13/06/2013.

[127] Xilinx, "7 Series FPGAs configurable logic block user guide," *User Guide, 2012,* [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf, Accessed on 25/11/2012.

[128] Xilinx, "Virtex-5 FPGA XtremeDSP design considerations," *User Guide, 2012,* [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug193.pdf, Accessed on 27/11/2012.

[129] Xilinx, "AccelDSP Synthesis Tool; User Guide," *Xilinx, 2008,* [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/acceldsp_user.pdf, Accessed on 10/06/2013.

[130] Xilinx, "System Generator for DSP; User Guide," *Xilinx, 2010,* [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/sysgen_user.pdf, Accessed on 10/06/2013.

[131] T. Feist, "Vivado design suite," *White Paper, 2012,* [Online]. Available: http://www.xilinx.com/support/documentation/white_papers/wp416-Vivado-Design-Suite.pdf, Accessed on 05/12/2012.

[132] C. M. T. Santiago T. Pérez, Jesús B. Alonso and José L. Vásquez (2011), "Design Methodology with System Generator in Simulink of a FHSS Transceiver on FPGA, Applications of MATLAB in Science and Engineering, Prof. Tadeusz Michalowski (Ed.), ISBN: 978-953-307-708-6, InTech, DOI: 10.5772/23812," [Online]. Available:: http://www.intechopen.com/books/applications-of-matlab-in-science-and-engineering/design-methodology-with-system-generator-in-simulink-of-a-fhss-transceiver-on-fpga, Accessed on 10/06/2013.

[133] M. Lema and O. Mitchell, "Absolute moment block truncation coding and its application to color images," *IEEE Transactions on Communications*, vol. 32, pp. 1148-1157, 1984.

[134] Xilinx, "Xilinx Co.," *Documentation, 2012,* [Online]. Available: http://www.xilinx.com/tools/sysgen.htm, Accessed on 05/06/2013.

[135] Altera, "Logic Elements and Logic Array Blocks in the Cyclone III Device Family," [Online]. Available: http://www.altera.co.uk/literature/hb/cyc3/cyc3_ciii51002.pdf, Accessed on 02/04/2013.

[136] S. McKeown and R. Woods, "Low power field programmable gate array implementation of fast digital signal processing algorithms: Characterisation and manipulation of data locality," *IET Computers and Digital Techniques*, vol. 5, pp. 136-144, 2011.

[137] R. Woods, McAllister, J., Turner, R., Yi, Y., Lightbody, G., "FPGA-based implementation of signal processing systems," *Wiley, New Jersey*, 2008.

[138] G. Shi, W. Liu, L. Zhang, and F. Li, "An efficient folded architecture for lifting-based discrete wavelet transform," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 56, pp. 290-294, 2009.

[139] C. Y. Xiong, J. W. Tian, and J. Liu, "A note on "flipping structure: An efficient VLSI architecture for lifting-based discrete wavelet transform"," *IEEE Transactions on Signal Processing*, vol. 54, pp. 1910-1916, 2006.

[140] C. T. Huang, P. C. Tseng, and L. G. Chen, "Flipping structure: an efficient VLSI architecture for lifting-based discrete wavelet transform," *IEEE Transactions on Signal Processing*, vol. 52, pp. 1080-1089, 2004.