Efficient VLSI Architectures for Image Compression Algorithms

Vijay Kumar Sharma

Department of Electronics & Communication Engineering
National Institute of Technology Rourkela
Rourkela-769008, Odisha, India

Efficient VLSI Architectures for Image Compression Algorithms

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

Master of Technology (Research)

In

Electronics and Communication Engineering

By

Vijay Kumar Sharma

(Roll No. 609EC101)

January 2012

Under the guidance of

Dr. U. C. Pati

Dr. K. K. Mahapatra

Department of Electronics & Communication Engineering
National Institute of Technology Rourkela
Rourkela-769008,Odisha, India

To all who have shown Love and Support

Department of Electronics & Communication Engineering
National Institute of Technology Rourkela
Rourkela-769008, Odisha, India

Certificate

This is to certify that the thesis entitled "***Efficient VLSI Architectures for Image Compression Algorithms***" by Mr. Vijay Kumar Sharma, submitted to the National Institute of Technology, Rourkela for the degree of Master of Technology (Research), is a record of an original research work carried out by him under our supervision in the department of Electronics and Communication Engineering during session 2009-2011. We believe that the thesis fulfills the part of the requirements for the award of degree of Master of Technology (Research). Neither this thesis nor any part of it has been submitted elsewhere for the degree or academic award.

Prof. U. C. Pati
Department of Electronics and Communication Engineering
National Institute of Technology, Rourkela

Prof. K. K. Mahapatra
Department of Electronics and Communication Engineering
National Institute of Technology, Rourkela

Acknowledgement

I am grateful to my research advisors Prof. U. C. Pati and Prof. K. K. Mahapatra for providing me the opportunity to realize this work. They inspired, motivated, encouraged and gave me full freedom to do my work with proper suggestions throughout my research work. I am indebted to Prof. K.K. Mahapatra for his kind and moral support throughout my academics at NIT Rourkela.

I am also grateful to NIT Rourkela for providing adequate facilities in my research work.

I acknowledge MHRD and DIT (Ministry of Information & Communication Technology), Govt. of India, for giving financial support.

My special thanks to Prof. S. K. Patra, Prof. D. P. Acharya and Prof. A. K. Swain for their help and valuable suggestions. I would like to thank Sushant sir, my senior, for helping me in VLSI.

I am thankful to Prof. S. Meher and Prof. S. K. Behera for his inspiration and support. I am also thankful to Prof. G. S. Rath, Prof. B. D. Sahoo, Prof. K. B. Mohanty and Prof. N.V. L. N. Murty for their valuable support in my research work.

I would like to give special thank to Prakash sir, Preeti madam and Ranjan sir who helped and supported me throughout my research work.

I must thank Karupannan sir, Kanhu sir, Jagannath and Tom sir for their support and help. I also thank Srinivas sir and Venkatratnam sir for their good attitude and behavior.

I sincerely thank to all my friends, Research scholars of ECE Department, M.Tech (VLSI) students and all academic and non-teaching staffs in NIT Rourkela who helped me.

Vijay Kumar Sharma

Contents

List of Figures

List of Tables

Symbols Used

Name	**Symbol**
Number	#

Abstract

An image, in its original form, contains huge amount of data which demands not only large amount of memory requirements for its storage but also causes inconvenient transmission over limited bandwidth channel. Image compression reduces the data from the image in either lossless or lossy way. While lossless image compression retrieves the original image data completely, it provides very low compression. Lossy compression techniques compress the image data in variable amount depending on the quality of image required for its use in particular application area. It is performed in steps such as image transformation, quantization and entropy coding. JPEG is one of the most used image compression standard which uses discrete cosine transform (DCT) to transform the image from spatial to frequency domain. An image contains low visual information in its high frequencies for which heavy quantization can be done in order to reduce the size in the transformed representation. Entropy coding follows to further reduce the redundancy in the transformed and quantized image data.

Real-time data processing requires high speed which makes dedicated hardware implementation most preferred choice. The hardware of a system is favored by its low-cost and low-power implementation. These two factors are also the most important requirements for the portable devices running on battery such as digital camera. Image transform requires very high computations and complete image compression system is realized through various intermediate steps between transform and final bit-streams. Intermediate stages require memory to store intermediate results. The cost and power of the design can be reduced both in efficient implementation of transforms and reduction/removal of intermediate stages by employing different techniques.

The proposed research work is focused on the efficient hardware implementation of transform based image compression algorithms by optimizing the architecture of the system. Distribute arithmetic (DA) is an efficient approach to implement digital signal processing algorithms. DA is realized by two different ways, one through storage of pre-computed values in ROMs and another without ROM requirements. ROM free DA is more efficient. For the image transform, architectures of one dimensional discrete Hartley transform (1-D DHT) and one dimensional DCT (1-D DCT) have been optimized using ROM free DA technique. Further, 2-D separable DHT (SDHT) and 2-D DCT

architectures have been implemented in row-column approach using two 1-D DHT and two 1-D DCT respectively.

A finite state machine (FSM) based architecture from DCT to quantization has been proposed using the modified quantization matrix in JPEG image compression which requires no memory in storage of quantization table and DCT coefficients. In addition, quantization is realized without use of multipliers that require more area and are power hungry.

For the entropy encoding, Huffman coding is hardware efficient than arithmetic coding. The use of Huffman code table further simplifies the implementation. The strategies have been used for the significant reduction of memory bits in storage of Huffman code table and the complete Huffman coding architecture encodes the transformed coefficients one bit per clock cycle.

Direct implementation algorithm of DCT has the advantage that it is free of transposition memory to store intermediate 1-D DCT. Although recursive algorithms have been a preferred method, these algorithms have low accuracy resulting in image quality degradation. A non-recursive equation for the direct computation of DCT coefficients have been proposed and implemented in both 0.18 μm ASIC library as well as FPGA. It can compute DCT coefficients in any order and all intermediate computations are free of fractions and hence very high image quality has been obtained in terms of PSNR. In addition, one multiplier and one register bit-width need to be changed for increasing the accuracy resulting in very low hardware overhead. The architecture implementation has been done to obtain zig-zag ordered DCT coefficients. The comparison results show that this implementation has less area in terms of gate counts and less power consumption than the existing DCT implementations. Using this architecture, the complete JPEG image compression system has been implemented which has Huffman coding module, one multiplier and one register as the only additional modules. The intermediate stages (DCT to Huffman encoding) are free of memory, hence efficient architecture is obtained.

Chapter 1

Introduction

An image in its original representation carries huge amount of data. Thus, it requires large amount of memory for storage [1]. Image compression is an important area in image processing which efficiently removes the visually insignificant data [2–8]. Compressed images are sent over limited bandwidth channel with some additional processing for robust (error free) transmission [9–12]. Transform based image compression algorithm is a most preferred choice which consists of image transform (in non-overlapping blocks), quantization of transformed coefficients and entropy coding [13]. Joint photographic expert group (JPEG) is a committee that standardizes the image compression algorithm [14]. The 8x8 block-wise two-dimensional discrete cosine transform (2-D DCT) is used as orthogonal transform in JPEG image compression [15]. Images compressed by this standard are used globally. This algorithm provides the user to choose between amount of compression and quality as per the requirement of the image in different applications. The variable amount of compression makes this algorithm very much suitable for the transmission purpose as user can adjust the bit rate of the transmission according to channel capacity.

JPEG is fixed algorithm and it has some flexibility that can be incorporated easily without any major changes in the basic structural feature [16–18]. JPEG system can be implemented in software as well as in hardware. Software solution is not promising for the applications requiring high speed. Therefore, real-time processing is done through the dedicated hardware [19,20]. In custom hardware implementation, architecture plays a vital role in deciding area, power and throughput of the design. Architecture optimizations lead to lower computational units (adders, multipliers), reduced memory size for storage of temporary variables and smaller interconnects. Architecture explorations to minimize the area and power consumption is a issue for portable devices running on battery. Low silicon area reduces the cost of the appliance [21,22] and low

power consumption increases the battery lifetime (time between recharges for chargeable battery) which in turn reduces the weight of the battery and overall size [23]. 2-D DCT is a complex algorithm and requires high computations. Further, subsequent stages in transform based image compression require high memory storage along with arithmetic circuits. For portable devices, having image compression system (like JPEG compression in digital camera [24–27]), low-cost design, that can be achieved by reducing silicon area is highly required [28–31]. By efficiently designing the hardware architecture, image compression can be performed with low-cost and low power budget.

1.1 Motivation

System level implementation of a digital device can be performed in embedded processors, digital signal processors (DSPs), application specific instruction set processors (ASIPs), reconfigurable logic/processors and dedicated hardware. Each implementation gives best performance for a particular application area. Unlike embedded processors, where low cost and low power consumption are basic requirements, price and performance of DSP processors vary according to application areas. They come in three categories, i.e., low cost, low power midrange and diversified high end. In the high end category, ultra high speed applications are implemented in DSPs [32]. ASIPs are designed for a particular application area. Their hardware and instruction-set may be optimized for power, area or performance. In terms of power consumption and hardware cost ASIPs are intermediate between general purpose processors (GPPs) and application specific integrated circuits (ASICs) [33–36].

Although processors, mentioned above, have flexibility that their functionality can be modified by changing the soft codes without any hardware modifications unless major changes (like throughput improvement by adding hardware in parallel or pipeline manner) are required, it (flexibility) comes at the cost of lower energy efficiency. Fig.1.1 depicts the energy efficiency of various implementations with respect to the flexibility. Reconfigurable processors provide the functionality of a hardware accelerator by assembling a number of functional units on temporal basis through configurable interconnect and configurable bus of the processor. Thus, when accelerator work is finished, the same functional units can be used for other applications, unlike dedicated

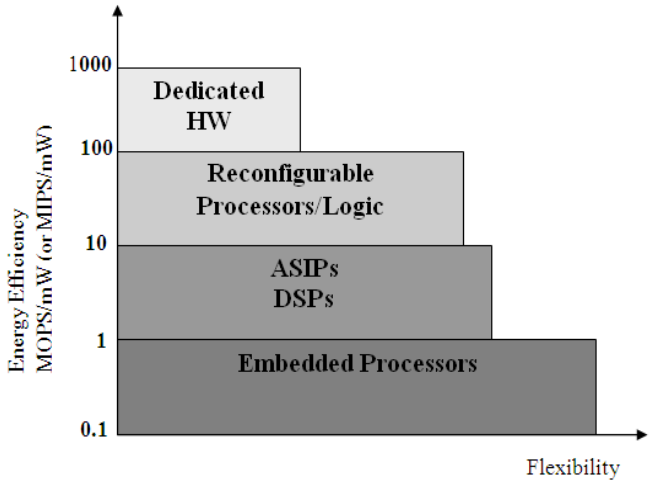


Fig.1.1 Energy efficiency on different implementations[23]

accelerator in DSPs which brings additional hardware overhead. FPGA is used for this purpose [23]. DSP processors are traditionally used inside the digital camera. Nevertheless, it requires the hardware processor and memory to store soft codes. Since the JPEG is a standard and changes in it is rare (and almost none), dedicated hardware for the JPEG compression in a digital camera is a promising solution because of the following reasons. Dedicated hardware possesses the maximum energy efficiency as compared to embedded processors, DSPs and reconfigurable hardwares, that increase the battery life time between recharges and has the low silicon area. It is important in camera (and in all consumer electronic appliances) because silicon area directly relates to cost of the device. In battery perspective, there is a constant annual growth of battery capacity in tune with the technology evolutions that enabled the battery volume shrinkage and also good talk-time for first WCDMA phones (Fig.1.2) [37]. But, for multimedia space (very high computation is involved) there is need to reduce power as battery capacity is not enough to meet the computations. Dedicated hardware has high speed which is necessary for the digital camera (and in all real time applications) where images are captured, compressed and stored in a pipeline manner in real time. Efficient design strategy of the dedicated hardware system can lead to reduction in power demanding computational units such as adders and multipliers [38]. An image processing system requires high storage (memory) elements. Moreover, memory related operations dominate the system

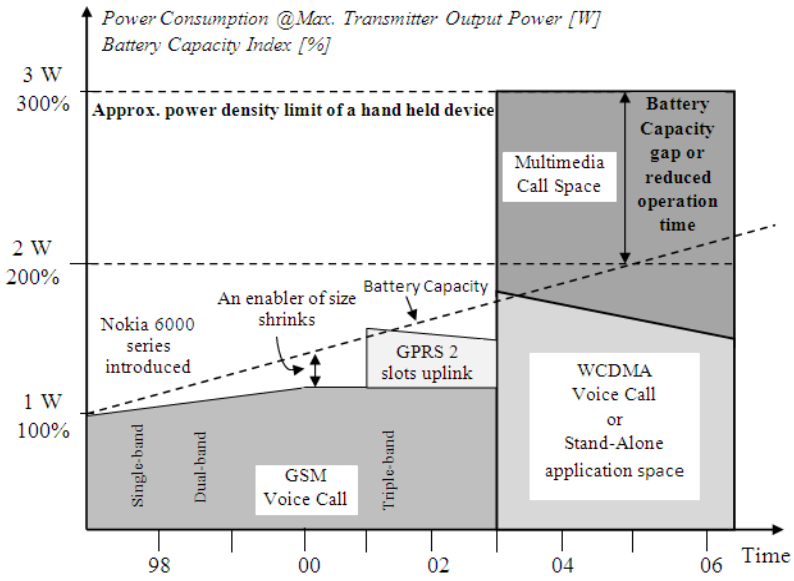


Fig.1.2 Trends in power consumption and battery capacity [37]

power consumption [39,40]. Techniques can be used for well defined system to reduce the memory and hence power consumption [41].

1.2 Background

Systolic architecture is used to implement digital signal processing and arithmetic algorithms in which fast processing is required. It was the preferred design approach for the special purpose systems because of its simple, regular and pipeline operations performed by set of small interconnected similar array cells called processing elements (PEs). Breaking the whole processing into small cells has the two major advantages. In case of systolic architecture, a set of data brought from the memory is processed by several PEs in a pipeline structure i.e., multiple operations are performed on each data item. Therefore, computation is increased at the same memory bandwidth [42]. One example for systolic array is given by H.T. Kung [42], where convolution operation were mapped on systolic array. A major area and power consuming module in VLSI is multiplier. Past research in the implementation of any DSP algorithm using systolic array was centered on the reduction of the number of multipliers. In many literatures, DCT is implemented by systolic array architectures with the purpose of reducing number of

multipliers [43–48]. Apart from systolic, other DCT implementations also favored the multiplier reduction. H. Malvar [49] implemented DCT using DHT with reduced number of multipliers. Y.-M. Chin *et al.* [50] proposed a new convolution based algorithm for computing DCT. Still, all these algorithms are not free of multipliers, i.e. they need multipliers along with adders and other logic components for the VLSI implementation. Distributed arithmetic (DA) algorithm can implement DSP algorithms without multipliers, which is important for area and power savings in VLSI designs. For DA implementation of inner product of arrays, one of the inputs should be constant array. ROM based DA relies on the manual pre-computations of constants and their storage in ROM. These pre-computed values are fetched from the ROM addressed by the input bits. According to S.A. White [51], by careful design one may reduce the total gate count in a signal processing arithmetic unit upto 80 percent. ROM based DA is utilized for DCT implementations in literatures [52–54]. ROM based design is not preferred choice in VLSI because ROM has slow speed (ROM access time) and more power consumption [55,56]. Moreover, size of memory (ROM) has to be increased exponentially with the size of transform and also high accuracy requirements. ROM free DA architecture exploits the sparse nature of matrix formed by binary representation of coefficients (most places are zero) in contrast to pre-computing and storing them in ROM based DA i.e., constant coefficients are distributed. Shams et. al, [57] first gave the analysis of ROM free DCT and named the algorithm NEDA (New DA). The reduced adder tree of ROM free DA is implemented by P. Chungan *et al.* [58]. Yuan-Ho Chen *et al.* [59] proposed high throughput DCT architecture using DA which uses less number of bit-width in DA precision for hardware reduction.

For 2-D DCT implementation from 1-D DCT using techniques mentioned above requires transposition memory resulting in high cost design along with irregular architecture for realization of data pipelined computation [60]. Also, row-column decomposition technique is unsuitable for the applications requiring transmission in limited bandwidth as all DCT coefficients need to be calculated in advance for sending, though it is sent one by one. To reduce the circuit cost, direct recursive computation of 2-D DCT is carried out using recursive kernel in which DCT coefficients are computed one by one at regular clock cycles [61–67]. The disadvantage of recursive kernel is that

accuracy is reduced to a large extent due to round-off error. More errors are introduced as recursive cycle increases because each processed register values requires higher number of bits for its representation and fixed size of register in VLSI makes it non-practical.

Quantization and Huffman coding are the other parts in the image compression system which requires huge storage and controlling circuitry. Quantization is division of 2-D DCT coefficients with a quantization step-size (different for different DCT coefficients in 8x8 blocks). Zig-zag ordering [15] has to be performed of the quantized DCT coefficients to encode the most important image information contents first. Hardware implementation of complete JPEG image compression is done in literatures [19], [68–71]. For the Huffman coding in JPEG compression, JPEG committee provides an optimized table (having codes for DCT coefficients) for the Huffman code. These tables need to be stored in memory. M. Kovac *et al.* [69], have implemented quantization with 16-bit multiplier and RAM. M. Kovac *et al.* [70] used 13x10-bit multiplier and RAM to store quantization table. Adders and shifters along with 64x12-bits ROM memory have been used by L. V. Agostini *et al.* [19]. M. Kovac *et al.* [70] used adders and shifters for division purpose and one division takes eight clock cycles. Five adders and five pipeline register stages along with quantization table have been used by Sung-Hsien Sun *et al.* [71] for quantization. The zig-zag ordering is performed by 8x8 arrays of register pairs by M. Kovac *et al.* [69] and M. Kovac *et al.* [70] whereas L. V. Agostini *et al.* [19] used time-interleaved RAM pairs of size 64x10-bits for the reading and writing operations. Similarly, Sung-Hsien Sun *et al.* [71] have used two RAM blocks for loading 64 DCT coefficients. Agostini *et al.* [19] used two ROM having sizes 12x13-bits and 176x21 bits to store Huffman code table. Efficient way of storing Huffman code table is presented by Sun *et al.* [71], where instead of storing 16-bits code word (required for the base code [1]), 8-bits width of memory size has been used and the whole system operate at 4.1 MHz in FPGA implementation.

After review of these articles, efficient hardware architectures for the image compression have been proposed. The architectures are optimized in all the stages for memory as well as datapath reductions by appropriate controlling circuitry and also by exploiting redundant nature of image in original representation. The objectives and outline of the work proposed in the thesis are presented in the following section.

1.3 Objective of the Thesis

A novel equation for the computation of 2-D DCT coefficients without use of transposition memory is proposed. The equation can compute DCT coefficients one by one in any order in a non-recursive way. All the internal computations are performed in integer format making hardware architecture highly accurate for DCT coefficient computation. The fractional cosine values are stored in a register which is multiplied with a multiplier at last stage only. Therefore, hardware overhead becomes negligibly small as only a multiplier and a register bit width needs to be changed for higher accuracy of DCT coefficients. From the proposed equation, VLSI architecture is implemented in both FPGA as well ASIC library. The implemented architecture has less area and low power consumption when compared to existing 2-D DCT implementations. From this implementation, an additional multiplier and a register are enough to get the quantized and zig-zag ordered DCT coefficients without extra memory requirements and at the same latency.

Hardware architecture for computation of 1-D DCT with reduced area and power using memory free DA approach is presented and implemented in FPGA as well ASIC library for area and power comparisons. The presented 1-D DCT architecture reduces the DA computational units from architecture presented by P. Chungan *et al.* [58] from seven to three with clock latency increased by 3 cycles. Image compression using separable discrete Hartley transform (SDHT) has been done and it is found that SDHT performs same as DCT at high compression. Hardware architecture for DHT using ROM free DA is proposed which has less adder bit-width requirement than DCT. The architecture for 1-D DHT and 2-D DHT is implemented in FPGA.

A simple finite state machine (FSM) based architecture for computation of DCT to zig-zag ordering of quantized DCT coefficients is proposed. The architecture removes memory requirements for 64 DCT coefficients storage, quantization table storage as well storage of quantized coefficients for zig-zag ordering. The energy compaction property of DCT coefficients is studied by reconstructing the image using less number of DCT coefficients.

Huffman coding is implemented for JPEG Huffman code table. Strategies have been used to reduce the code memory requirements.

The major research works done are listed here:

1. The energy compaction property of DCT is studied by reconstructing the different images with the help of only selected DCT coefficients in the decompression.
2. Image compression and decompression is done using 2-D separable discrete Hartley transform (2-D SDHT). Basis function image of SDHT is plotted, PSNR vs. compression ratio (rate-distortion curve) is plotted and compared with the 2-D DCT for different standard images.
3. FPGA implementation of SDHT is performed using efficient ROM free DA approach and results are compared with ROM based DA.
4. Area and power efficient VLSI architecture for 8 point 1-D DCT using ROM free DA is presented and implemented in FPGA as well as ASIC library.
5. A simple finite state machine (FSM) based VLSI architecture from DCT to Zig-zag reordering of transformed coefficients for JPEG baseline encoder using quantization table suitable for less complex hardware design is presented and implemented in FPGA as well as standard cell.
6. Non-recursive equation and its VLSI architecture for direct computation of 8x8 two dimensional 2-D DCT without transposition memory for high image quality is presented and implemented in FPGA as well as standard cell based technology.
7. Huffman coding is implemented using memory requiring less number of bit storage as compared to original.

1.4 Chapter Wise Contribution of the Thesis

Chapter-1 : Introduction

Introduction to transform based image compression along with motivation behind the dedicated hardware design for image compression is presented. Background work and main research contribution is also mentioned.

Chapter-2 : Image Compression

The focus of this chapter is to study the energy compaction property of DCT and DHT in transformed based image compression. Image reconstruction is done using selective 2-D DCT coefficients. Quality assessment is done by reconstruction. Compression ratios are tabulated for different images at different quantization level. 2-D SDHT is used for the image compression and decompression.

Chapter-3 : Distributed Arithmetic and its VLSI Architecture

Distributed Arithmetic (DA) implementation approach for DCT and DHT is main aim of this chapter. Efficient implementation of DCT and DHT using ROM free DA approach is described. Different implementation results (like 1-D DCT, 1-D DHT etc.) are summarized and compared with the existing implementations.

Chapter-4 : Efficient JPEG Image Compression Architecture

This chapter focuses on efficient architecture of JPEG from DCT to zig-zag ordering where memory requirements for intermediate storage of DCT coefficients before and after quantization is removed by simple control circuit design. Also, efficient implementation of Huffman code table with reduced storage is done.

Chapter-5 : Direct Computation of 8x8 2-D DCT Coefficients Equation and Its Hardware Architecture

The direct computation equation of 2-D DCT coefficients is explained. The proposed equation computes DCT coefficients in non-recursive way without transposition memory and require less hardware for image compression as demonstrated with the complete JPEG implementation.

Chapter-6 : Summary and Conclusions

The comprehensive summary of the thesis is provided with scope for future work.

1.5 Summary

In this introductory chapter, transformed based image compression and requirement for the low cost and low power image compression hardware are introduced. Motivations

towards the dedicated hardware implementation rather than software implementation are explained. The related works done in hardware implementation of DCT and complete JPEG image compression system is highlighted. The thesis objective is mentioned with major contributions illustrated point wise. Finally, chapter organization of the thesis is summarized.

Chapter 2

Image Compression

2.1 Introduction

A digital image is two-dimensional functional in space where amplitudes at each location are called pixels. There are different types of images depending upon the different number of data bits per pixel for their representation. Quality of an image can be assessed either visually or by mathematical formulation. The former is called subjective quality assessment and the later objective quality assessment. A common objective quality assessment metric for images obtained after decompression is PSNR (peak signal-to-noise ratio). Transform based lossy image compression is flexible as it can compress images at different qualities depending upon the application of the image. JPEG uses 8x8 block-wise 2-D DCT as the transform. DCT has very high energy compaction and its performance is almost similar to optimal Karhunen-Lo'eve transform (KLT) with the advantage of constant kernel and less computational complexity. Still, for the hardware implementation, similar kind of transform which will have less computational complexity and hence less hardware requirement with performance almost similar to DCT can be a preferred choice.

In this chapter, introduction about digital image representations and its various classifications have been given. Image quality assessment has been briefed. Basics of transform based image compression along with JPEG image compression have been described. The energy compaction property of DCT has been studied by compressing the images with few low frequency DCT coefficients. Compression ratios at different scale of compression for different standard images have been tabulated and images are displayed for quality visual assessment. Separable discrete Hartley transform (SDHT) has been introduced for image compression and decompression. Quality of images obtained from compression and decompression by SDHT is compared with DCT.

2.2 Image representation and classification

Image of a natural scene has infinite level of brightness and color intensity variations. Apart from intensity, they are continuous function in two dimensional space. To process the image for various applications by digital processors along with its storage in memory, image data obtained from electronic image sensors (CCD or CMOS) in digital camera, scanner or any similar device are converted into digital form by A/D converter. Sampling and quantization steps are used [1]. The infinite intensity levels of the image has now become digital having finite levels. Spatial continuity, itself being sampled by the fixed points present on the sensor, is converted to discrete. Continuous image signal (natural scene), now, is a two dimensional digital function, represented by $f(x, y)$, where the magnitude of function f represents the intensity from among finite levels of intensities at any point (x, y) in the space. The coordinate (x, y) is discrete as shown in Fig.2.1. The intensities at different points in space are called pixel elements or pixels of the image. One example of finite level of intensities can be all integral values from 0 to 255. In general, any digital image will have the fixed number of pixel elements in horizontal as well as vertical directions. The term *size* of the image is used for the total number of pixel elements in an image. It is represented by MxN, where M is the number of rows and N is the number of columns of image data.

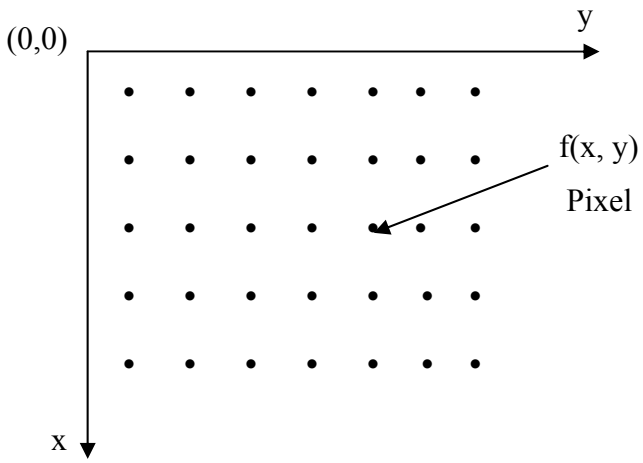


Fig.2.1 Representation of digital image in two dimensional spatial coordinate

In digital representation, the magnitude of intensity is represented by a fixed number of bits for the entire pixels. Classification of image on the basis of the number of bits used for representing each of its pixel value is as follows [72]

(a) Bi-level image

Each pixel will have one bit (binary) value, representing black and white. Textual information can be represented by the bi-level image.

(b) Grayscale image

This is a most common type of image used in many applications. A grayscale image represents the 2^n shades of a gray, where n is the number of bits representing each pixel. The 8-bits (one byte) representation is most preferred and used for display in computer monitor and printing purpose as well. In 8-bit representation there are 256 shades of gray (or intensities) between black and white.

(c) Continuous-tone image

In a continuous-tone image there are many shades of a color (or gray). In other words, one pixel has many intensity levels such that nearby pixel intensity, though it differs by one unit intensity level, appears same to the eyes. Images obtained from the digital cameras and scanners are example of continuous-tone image. Color image is represented by 24-bits pixel value in three color component planes R (red), G (green) and B (blue) with 8-bits allocated for intensities of each color.

2.3 Image Quality Measurement Metric

Images are degraded in quality while going through the different processing steps such as acquisition, compression, transmission and reproduction [73]. In image and video processing fields, there are various systems and they all convey visual information for human perception. There is trade-off between system resources and visual quality obtained from these systems [74]. These requirements lead to necessity for the image quality measurement metric. In addition, there are engineers working on the optimization of signal processing algorithms [75]. So, it is also important for this perspective to test

algorithms for quality of the obtained signal (benchmarking algorithms). There are two methods to evaluate the quality of images. These are as follows:

- Subjective Quality Measure

In the most of the image processing applications, human beings are the ultimate viewer and hence, subjective quality evaluation is done by the human beings. Consensus of the individuals regarding quality of compressed/decompressed images is taken in account. Viewer gives ratings among different choices available. Mean opinion score (MOS) is a numerical value that is the output of the observation given by [13],

$$MOS = \frac{\sum_{i=1}^{C} n_i R_i}{\sum_{i=1}^{C} n_i} \tag{2.1}$$

Here, R_i is the numerical value corresponding to category i, n_i is the number of judgments in that category and C is the number of category. However, subjective evaluation is expensive and process is slow. So, quality measurement cannot be incorporated in the automatic systems [73, 74].

- Objective Quality Measure

Accurate and automatic quality measurement can be done by formulating a mathematical model. Signal-to-noise ratio (SNR) is a simple mathematical model of quality measurement expressed in terms of mean square error (MSE) and signal variance (σ_s), given by [13],

$$SNR(dB) = 10\log_{10}\left(\frac{\sigma_s^{\,2}}{MSE}\right) \tag{2.2}$$

where variance is expressed by,

$$\sigma_s^{\,2} = \frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N}(f[m,n]-\mu)^2, \quad \mu = \frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N} f[m,n] \tag{2.3}$$

and MSE is given by,

$$MSE = \frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N}(f[m,n] - \hat{f}[m,n])^2 \quad (2.4)$$

where $f[m, n]$ represents original image and $\hat{f}[m, n]$ represents the image after the application of compression/decompression process, the quality of which has to be determined. The size of the image is MxN. SNR being dependent on the image variance, another quantitative measurement metric is peak signal-to-noise ratio (PSNR). PSNR is expressed by the same SNR equation with variance replaced by maximum intensity level in the image representation. In case of 8-bits per pixel (gray scale) image, the maximum intensity is 255.

$$PSNR(dB) = 10\log_{10}\left(\frac{255^2}{MSE}\right) \quad (2.5)$$

PSNR is a better measurement matrix for comparing two images processed by the hardware, as it gives the truncation error introduced due to limited bit-width representation (e.g., register bit-width) [57].

2.4 Image Compression Model

Image compression reduces the amount of data from the original image representation. There are two approaches to compress an image. These are:

(a) Lossless compression
(b) Lossy compression

Image data compressed by lossless compression method can be retrieved back accurately in the reverse process called decompression. Lossless compression method has the disadvantage that images can be compressed by a maximum compression ratio of about 3 to 4 (very low compression), where compression ratio (CR) is given by,

$$CR = \frac{n1}{n2} \quad (2.7)$$

Here, n1 is the total number of bits in original image and n2 is the total number of bits in compressed image. In lossy compression method, an image is compressed at the cost of

removing unwanted information from it which cannot be perceived by human visual system (the human eyes are not able to distinguish the changes). With the help of lossy compression technique, images can be compressed to a large extent (very high compression) subject to quality requirement for image application. Hence, lossy compression is a most common and used in many image and video coding standard such as JPEG, MPEG etc.

Fig.2.2 shows a general image compression model. Image data representation has redundancy (also called pixel correlation, interpixel redundancy or spatial redundancy), in the sense, a pixel value can be predicted by its neighborhood pixels [1, 76]. De-correlation process removes the spatial redundancy and hence, facilitates compression. Some of the techniques used for this process are predictive coding, transform coding and subband coding [76]. Apart from the interpixel redundancy, there is statistical redundancy present in the data after de-correlation (not only image but any data possess statistical redundancy). This is removed by entropy encoding process where more probable symbol is assigned less number of bits and vice-versa (also called variable length encoding). Huffman coding and arithmetic coding are two important techniques used for entropy encoding of data [77], [78]. Although, arithmetic encoding gives slightly

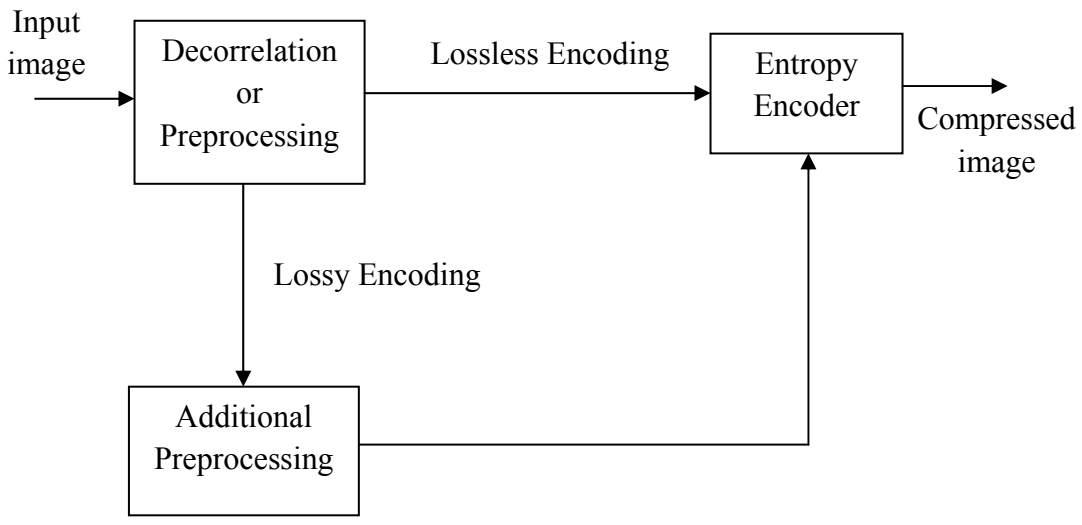


Fig.2.2 A generalized image compression model [76]

more compression than the Huffman encoding, it is a more complex and computation intensive. Therefore, Huffman coding is preferred choice in hardware implementation of entropy coding. In case of lossless compression, images undergo entropy encoding directly after de-correlation, whereas lossy compression require additional preprocessing stage called quantization before it is encoded by entropy process. Quantization is irreversible process and it is the only lossy stage in image compression model.

2.5 Transform based Image Coding

Transform based image coding is most preferred and widely used lossy image compression (coding) method. Fig.2.3 shows the block diagram of transformed based image compression coding technique. The purpose of the transform is to remove interpixel redundancy (or de-correlate) from the original image representation. The image data is transformed to a new representation where average values of transformed data are smaller than the original form. This way the compression is achieved. The higher the correlation among the image pixels, the better is the compression ratio achieved. An image transform should have the following properties.

(a) Inverse transformation should exist
(b) De-correlate the original image data
(c) Clear separation of frequency

Inverse transformation is a pre-requisite requirement in any transform because transformed data should be re-constructed for image formation by inverse process (decompression). Orthogonal transform (like DCT, DHT, DWT, etc.) is used for this purpose. A de-correlation property makes the transformed data independent from each other. In lossy image compression, some coefficients are quantized to zero or altered to a

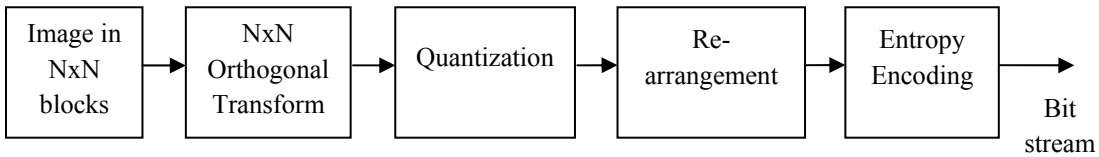


Fig.2.3 Transform based image compression model

new smaller value. Therefore, by the de-correlation property, in inverse transforms, original image data remains nearly unchanged. Frequency separation brings the transformed data made up of different frequency coefficients. An image contains very high visual information in the low frequency contents than the high frequency content. Very fine details are represented by high frequency contents of the image and in many applications, fine details are not required (also in many cases these details are not important as they are not visible to human eyes). Therefore, if clear order of frequency is known, high frequency coefficients can be ignored (quantized to zero) in the coding stage and hence compression is achieved. An ideal image transform should possess the following two properties. These are:

(a) Maximum energy compaction
(b) Less computational complexity

By the energy compaction, very few coefficients can have high values in the transform domain. Therefore, lesser the coefficients value, higher is the compression. Fast image compression is required in many compression systems and complex transform leads to high computation time making the process slower. Also, in case of faster implementation, dedicated hardware is used. Furthermore, high complex algorithm requires more hardware area, making the encoder design costly and also more power consuming.

Block based transform

Images can be transformed in non-overlapped smaller block size, like 4x4, 8x8, 16x16, 32x32, etc. Since, nearby pixels possess correlations, this trend is valid throughout the entire image pixels. Therefore, larger the block size taken for the transform, more correlation can be exploited and interpixel redundancy can be removed. In practice, it is found that average coding gain improvement is much lower when increasing the transform size [13, 79]. In contrast, the computational complexity increases by a larger amount with increase of block size. Hence, a compromise is made for the transform size between the computational complexity and coding gain and 8x8 block based transform is adopted in many image and video coding standards, like JPEG image compression, MPEG video compression, etc.

Quantization

Quantization process brings variable compression in the image compression process. Quantization process is done by dividing each transformed coefficients by a constant step size. In this process, transformed coefficients are made smaller to bring the compression [15]. Inverse quantization step restores original coefficients but with round-off error. Higher step size used for the quantization makes transformed coefficients smaller i.e., more compressed, with the cost of losing data in rounding and truncation. Quantization step size can be controlled by a variable number. Automatic compression size is incorporated by meeting the trade-off between image quality, bit-rate for transmission and memory size used for the storage of compressed images. Transformed coefficients possess the different frequency values. Accordingly, high frequency coefficients which carry low visual information can be quantized heavily without losing important image information.

Coefficients Re-arrangement

Many high frequency transformed coefficients when quantized heavily become zero. In order to code the quantize coefficients efficiently, Run-Length encoding procedure is applied. In Run-Length encoding procedure, not all zeros are encoded by a unique code for zero, but it is encoded by the number of zeros preceding the non-zero coefficient. Hence, larger zeros make little change in code length. Therefore, it is important that zero quantized coefficients are not distributed among non-zero coefficients, rather for optimal encoding, all zeros should come together in sequence. For this purpose, transformed and quantized coefficients are re-arranged in increasing frequency order so that higher frequencies (which are quantized to zero) appear last.

2.6 JPEG baseline Image Coding

JPEG baseline image coding is a transform based lossy image compression technique and it is standardized by JPEG committee [14]. Image is processed in 8x8 blocks to reduce the computational complexity for the implementation. The 8x8 block-wise 2-D DCT is taken followed by quantization of DCT coefficients. A typical quantization matrix is given by,

$$Q_m = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Quantized DCT coefficients are rearranged in increasing frequency order (zig-zag order) as shown in Fig 2.4(a) so as to encode the visually significant coefficients first. The first DCT coefficient is having zero frequency. It is called DC coefficient and the rest of the 63 coefficients are called AC coefficient [15]. DC coefficient from the previous block are subtracted with the current block (differential coding) and are encoded using Huffman coding. The DC coefficients represent the average image information of the block. The DC differential coding is performed to reduce the code size as nearest block possess the almost same average energy [1]. The AC coefficients are first encoded by run-length coding where an AC coefficient and runs of zero preceding this coefficient are grouped. This is performed because most of the high frequency coefficients (residing in bottom right region) become zero after quantization and hence efficient (short) binary code is obtained. The run-length coded data are then encoded by Huffman coding procedure. The JPEG committee provides a standard table for quantization as well as Huffman coding (Fig.2.4(b)). Quantization levels are stored in quantization table whereas, Huffman table contains the base codes of the AC and DC coefficients. For getting base code for a coefficient, its category (it is assigned for a range of coefficients [1]) and run-length code (for AC coefficients) form the address to fetch the base code from the table. Base code is extended with binary code of the coefficient to make the complete code of the coefficient. The DC and AC coefficients code are then combined to form bit-stream. The run of zeros more than 16 are encoded by special code. At the end when all coefficients in a block are encoded, a special code indicating end of block is inserted. Huffman coding can be performed without use of table, but it makes the encoding slower and also the computation more complex.

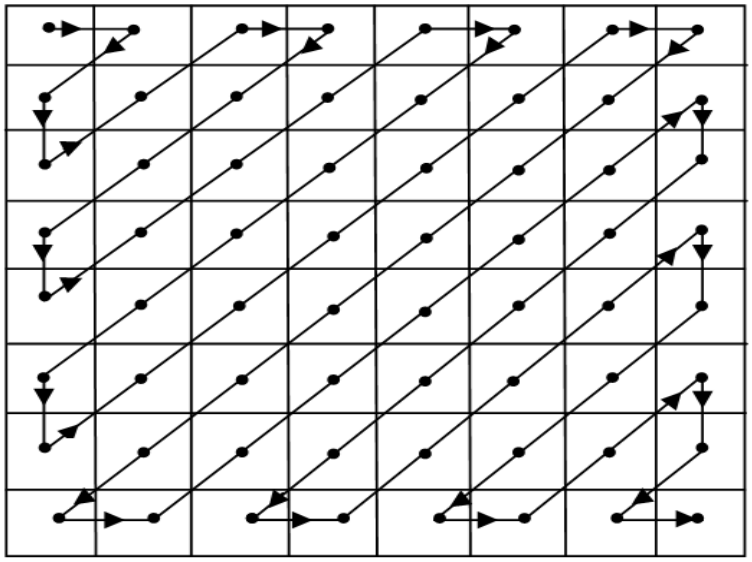


Fig.2.4(a) Zig-zag ordering for DCT coefficients

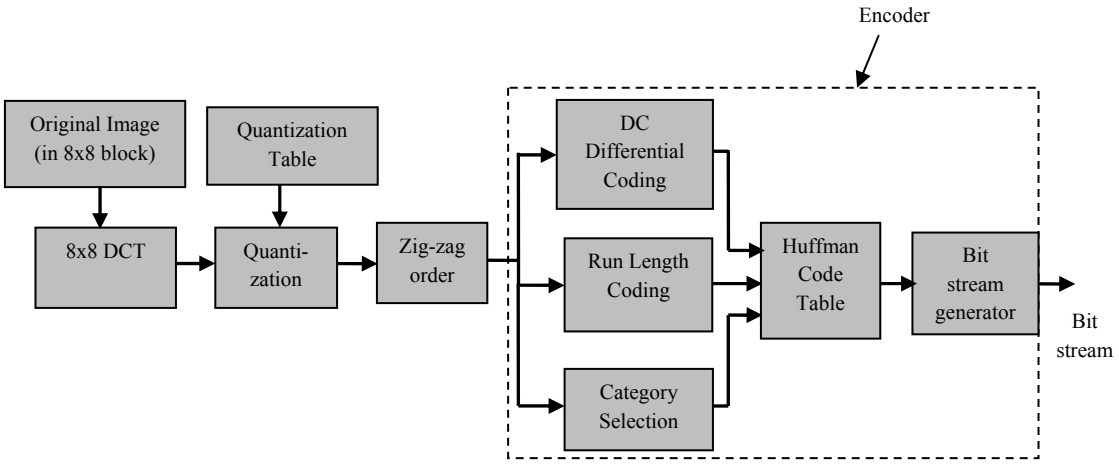


Fig.2.4(b) JPEG baseline Image compression

2.7 Discrete Cosine Transform (DCT)

DCT is an orthogonal transform. Karhunen-Lo'eve transform (KLT) is optimal in class of orthogonal transforms like Fourier transform, Walsh-Hadamard transform and Haar transform and has the best energy compaction [72, 79]. However, KLT is not ideal for practical image compression as its basis vectors has to be calculated according to the pixel values of the image (i.e., KLT is a data dependent). For each image, there will be

separate basis vectors that also need to be included in the compressed image for the decompression process. It was found that DCT performs close to KLT and their performances are also close with respect to rate-distortion criterion (quality at different compression) [79]. In addition, there are several fast and hardware efficient algorithms available for the computation of DCT [80–87]. Therefore, DCT became the widely used transform for lossy image encoding/compression and also in the several other signal processing applications.

2.7.1 2-D DCT Equation

For a NxN 2-D data $X(i, j)$, $0 \le i \le N-1$ and $0 \le j \le N-1$, NxN 2-D DCT is given by [64],

$$F(u,v) = \frac{2}{N} C(u)C(v) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} X(i,j) \times \cos\left(\frac{(2i+1)u\pi}{2N}\right) \cos\left(\frac{(2j+1)v\pi}{2N}\right) \tag{2.8}$$

where, $0 \le u \le N-1$ and $0 \le v \le N-1$ and C(u), C(v) = $1/\sqrt{2}$ for u, v=0, C(u), C(v) =1 otherwise. The 2-D DCT equation is separable transform and can be evaluated by first taking the 1-D DCT to rows followed by 1-D DCT to columns, where 1-D DCT is given by,

$$F(u) = \sqrt{\frac{2}{N}} C(u) \sum_{i=0}^{N-1} X(i) \cos\left(\frac{(2i+1)u\pi}{2N}\right) \tag{2.9}$$

With, C(u) defined as above. Fig.2.5 shows the 2-D DCT calculation from 1-D DCT using separable property. DCT transforms the spatial data into frequency domain.

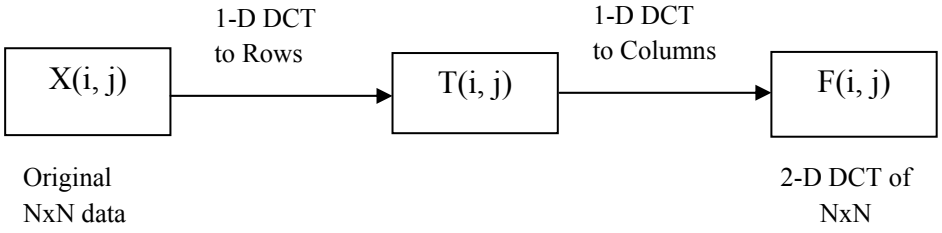


Fig.2.5 2-D DCT from separable property

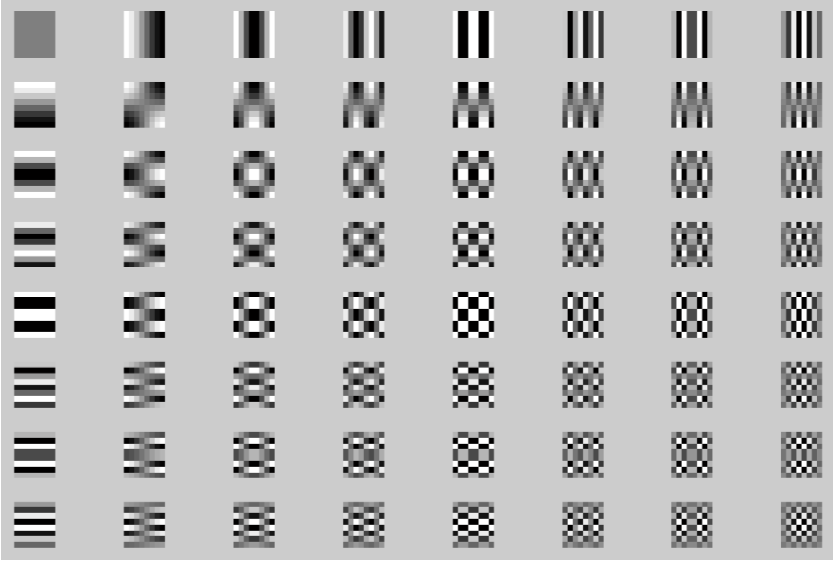
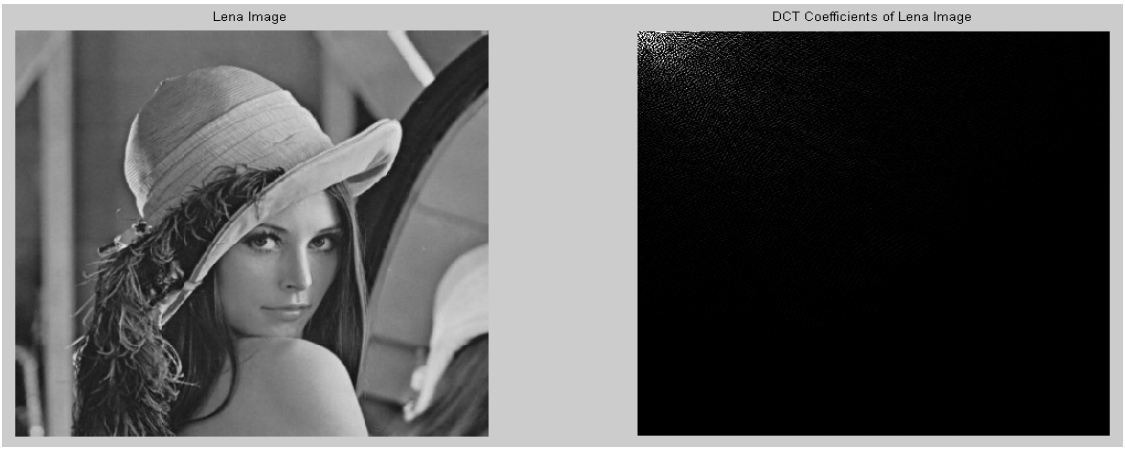


Fig.2.6 64 basis functions image of an 8x8 2-D DCT matrix

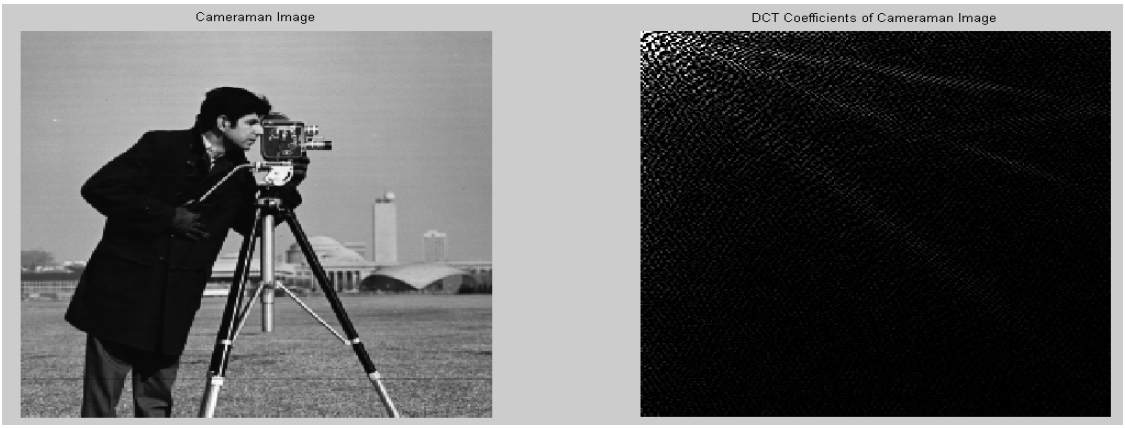
For *i, j*=0, cosine term will be zero and F(0,0) will represent the average value (DC) of all NxN pixels. Basis function images are generated for 8x8 2-D DCT as shown in Fig.2.6. Top-left image has no intensity variation and hence, it corresponds to DC frequency. Other 63 images are varying in intensity and shows spatial frequencies. Frequencies are increasing from top to bottom and left to right with bottom right representing the maximum frequency. Therefore, top left coefficient of any transformed image block corresponding to zero frequency is called DC coefficient and rest are called AC coefficients.

2.7.2 Energy Compaction Property of 2-D DCT

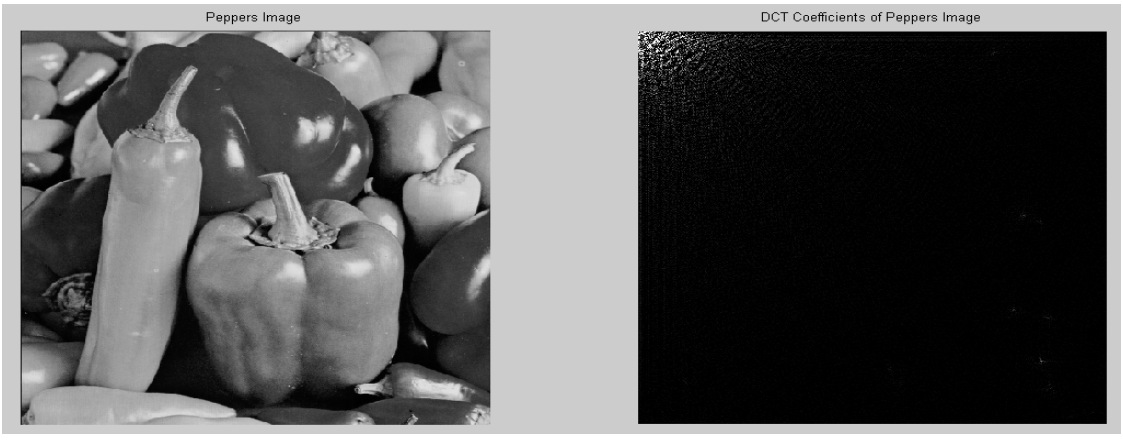
DCT has very good energy compaction. Most of the image energy is stored in few DCT coefficients. Images are transformed into 2-D DCT and images of coefficients are displayed in right side for three types of standard images as shown in Fig. 2.7. Top left side is brighter indicating high intensity, i.e., high numerical value of coefficients, whereas, rest of the parts are black that means they have almost zero value (and hence zero energy) as energy is proportional to square of the image intensity.



(a)



(b)

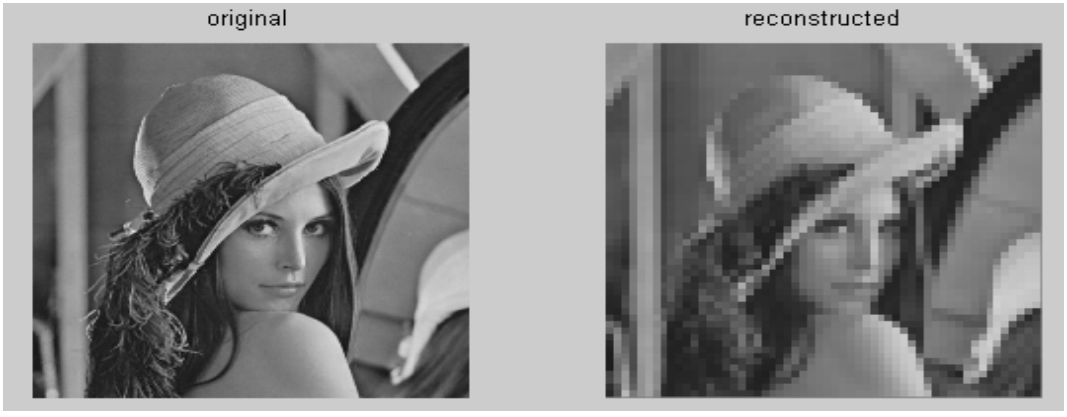


(c)

Fig.2.7 Energy compaction of DCT. Image (left) and its DCT coefficients' image (right) (a) 450x450 Lena, (b) 256x256 Cameraman and (c) 512x512 Peppers

2.7.3 Image Reconstruction by selective DCT coefficients

DC coefficient of DCT contains the average pixel values of the image. This is true for the block based transform as well. In case of block based transform, DC coefficients of each block carry most of the signal energy of that block and therefore, DC coefficients of the image have highest energy as compared to the average energy possess by total AC coefficients of entire blocks. This is shown in Fig.2.8, where Lena and Peppers images are first DCT transformed in 8x8 blocks. Then, AC coefficients of each block is discarded (quantized to zero) and image is reconstructed by Inverse DCT (IDCT) with the help of only DC coefficients of each block. Energy compaction property of DCT



(a)

(b)

Fig.2.8 Original (left) and reconstructed (right) image after quantizing all AC coefficients of 8x8 DCT to zero (a) Lena and (b) Peppers

coefficients discussed in Sub Section 2.4.2 clarify that most of the image energy is contained in few low order DCT coefficients. This observation can be exploited to reduce the computation of DCT in both hardware and software implementations. Four types of images are JPEG compressed and decompressed in three cases by selectively taking 8x8 DCT coefficients.

Case 1: All 64 DCT coefficients are taken for reconstruction.
Case 2: Only first row and first column DCT coefficients are taken for reconstruction.
Case 3: Only first 15 DCT coefficients in zig-zag ordered are taken for reconstruction.

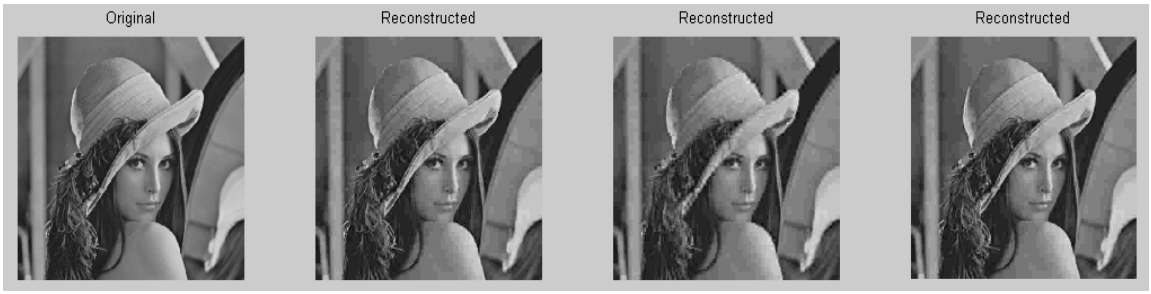
TABLE 2.1 shows the percentage improvement in compression ratio (CR) in case 2 and case 3 with respect to case 1. Shadow rows shows the CR for heavy quantization (higher value of quantization parameter i.e., "quality"). Fig.2.9 shows the reconstructed

TABLE 2.1
COMPRESSION RATIO OBTAINED FOR DIFFERENT QUANTIZATION LEVEL

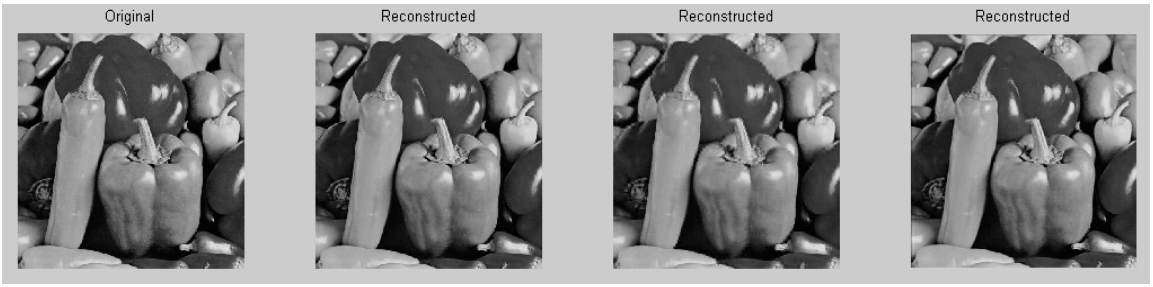
Images		Compression ratio in case 1 (all 64 DCT coefficients taken)	Compression ratio in case 2 (first row and first column DCT coefficients taken)	Compression ratio in case 3 (first 15 coefficients taken)	% improvement in compression ratio in case 2 (as compared to case 1)	% improvement in compression ratio in case 3 (as compared to case 1)
Lena (448x448)	quality=1	12.66	19.08	14.16	50.7 %	11.84 %
	quality=5	33.74	39.96	34.09	18.4 %	1.0 %
Peppers (512x512)	quality=1	12.50	18.17	14.02	45.3 %	12.1 %
	quality=8	41.50	45.24	41.58	9.0 %	0.1 %
Crowd (512x512)	quality=1	6.88	11.61	7.96	68.7 %	15.7 %
	quality=5	17.48	23.87	17.62	36.5 %	0.8 %
Cameraman (256x256)	quality=1	9.64	16.58	13.30	72 %	37.9 %
	quality=3	19.42	27.98	22.06	44.0 %	13.6 %

Note: "quality" is a parameter in JPEG compression which decides DCT coefficients quantization level

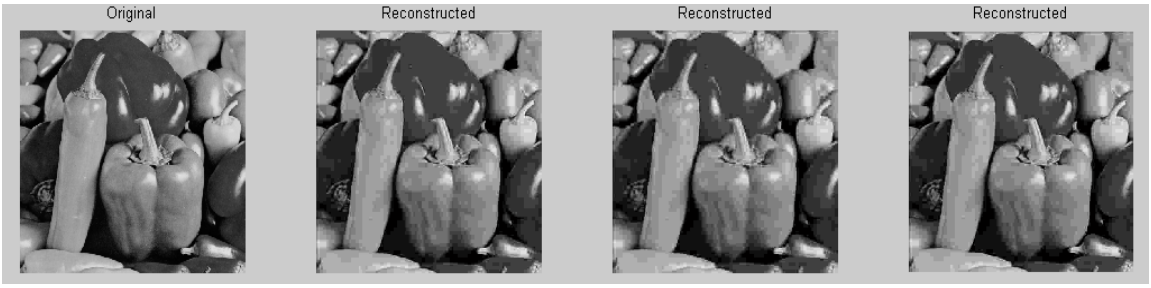
(a)



(b)



(c)

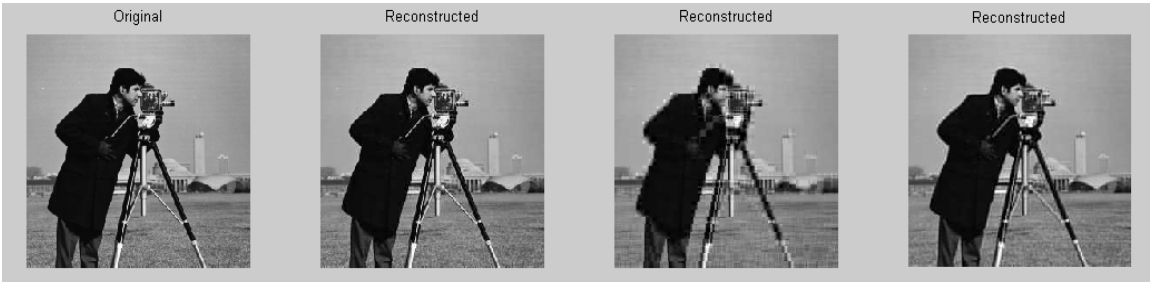


(d)

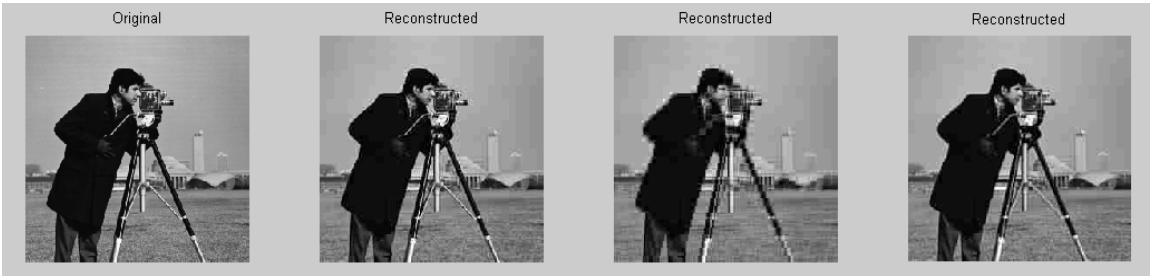
Fig.2.9 From left to right, original image, reconstructed image by taking all DCT coefficients, reconstructed image by taking first row and first column DCT coefficients, reconstructed image by taking first 15 coefficients in zig-zag order of (a) 448x448 Lena, quality=1, (b) 448x448 Lena, quality=5, (c) 512x512 Peppers, quality=1, (d) 512x512 Peppers, quality=8

(e)

(f)



(g)



(h)

Fig.2.9 (continued), (e) 512x512 Crowd, quality=1, (f) 512x512 Crowd, quality=5, (g) 256x256 Cameraman, quality=1, (h) 256x256, Cameraman, quality=3.

images in above mentioned three cases. Reconstructed images are shown for the quality comparison as psychovisual information (not visible to eyes [88]) are removed by

discarding the high frequency coefficients. The following observations can be made from TABLE 2.1 and Fig. 2.9.

- There is much improvement in compression ratio in case of low quantization without visual image quality degradation.
- At the higher quantization, smooth image (like Peppers, Lena) shows little improvement in compression ratio while detailed images (Cameraman) show still high improvement without visual quality degradation when first 15 low frequency coefficients are taken for image reconstruction, quantizing rest to zero.

From these observations, it can be concluded that DCT algorithm, which computes DCT coefficients one by one sequentially, can be made computationally efficient (at the same time low energy consuming) and faster by selectively taking the DCT coefficients for the image quality requirements. In addition, higher compression can be achieved by doing so.

2.8 Separable Discrete Hartley Transform (SDHT)

Discrete Hartley transform (DHT) has many applications in signal processing and communications [89–91]. Discrete Hartley transform (DHT) has been used as a substitute for discrete Fourier transform (DFT) by Bracewell *et al.* [92]. DHT is used in JPEG based image compression with DHT replacing the DCT by Pattanaik *et al.* [93]. Original 2-D DHT equation is not separable like 2-D DCT. In the literature [94], the concept of separable DHT was introduced by Watson. Separable 2-D DHT (SDHT) can be implemented in hardware by row-column transformation method of 1-D DHT and has very low hardware requirement as compared to DCT (shown in following chapter). Non separable DHT cannot be implemented using 1-D DHT making its implementation computation intensive. Separable 2-D DHT is given by the equation [94],

$$Y(u,v) = \sum_{x=0}^{N-1}\sum_{y=0}^{M-1} f(x,y)cas\left(\frac{2\pi ux}{N}\right)cas\left(\frac{2\pi vy}{M}\right) \tag{2.10}$$

where, $cas(x) = \cos(x) + \sin(x)$

and the 1-D DHT is given by the equation,

$$Y(k) = \sum_{n=0}^{N-1} X(n) cas\left(\frac{2\pi}{N} nk\right), \quad k = 0,1,...,N-1 \tag{2.11}$$

where, $H_{nk} = cas\left(\frac{2\pi}{N} nk\right)$ is the transform's kernel.

Fig. 2.10 shows the basis function image of SDHT. Higher frequencies coefficients occupy the middle place while lower are at boundaries. SDHT is tested for the performance in terms of PSNR with respect to DCT with two standard images in image compression area namely Lena and Cameraman. Images are compressed by JPEG standard principle with two modifications. In case of SDHT based transform, DCT block is replaced by SDHT and quantization matrix has been modified to quantize the SDHT coefficients as per the DCT (same numbers are used but in appropriate places). Fig.2.11 shows the performance curves. It can be observed that SDHT performs better in heavy quantization (at higher compression) while DCT performs better in lower compression in

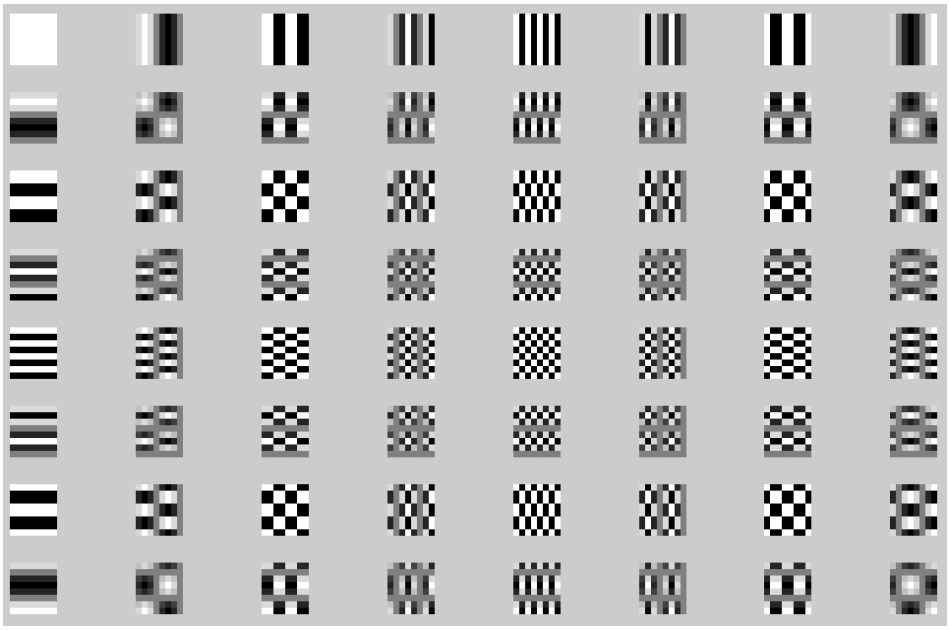
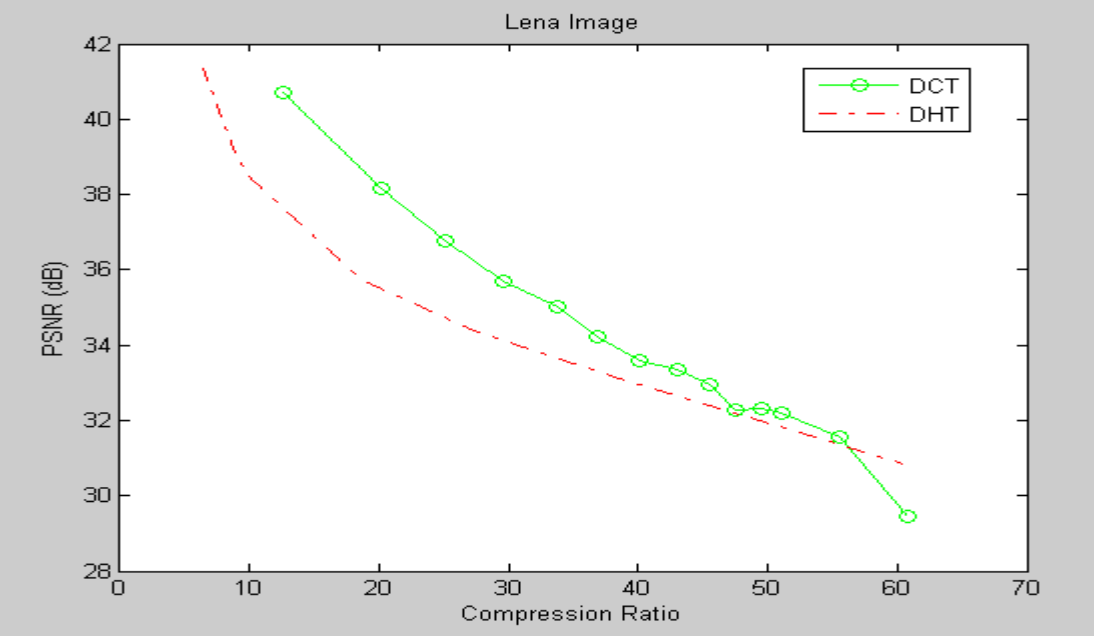
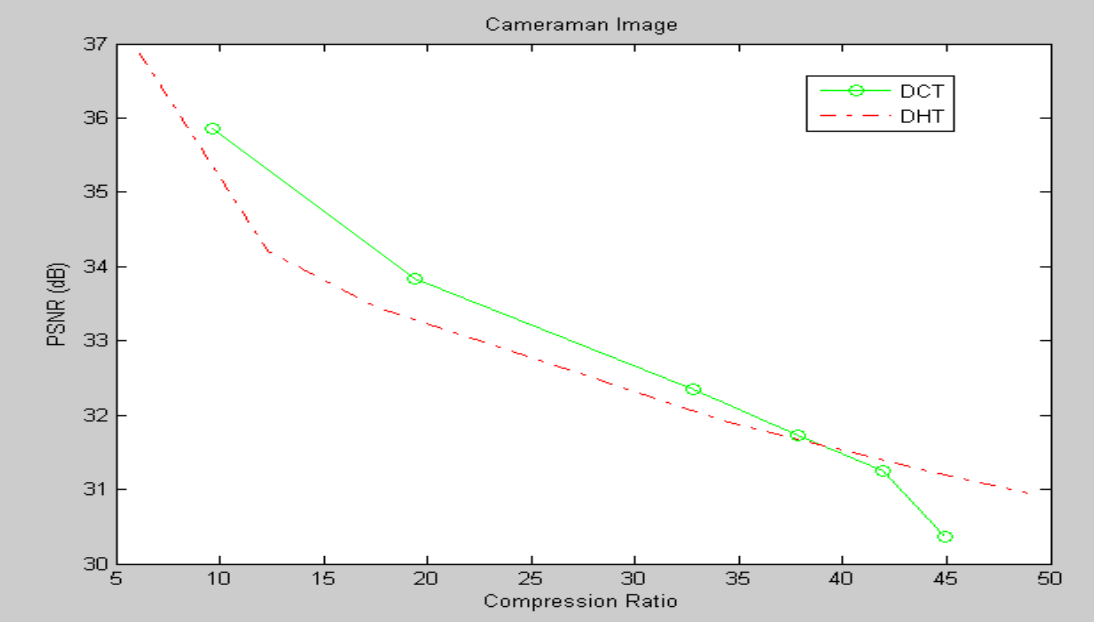


Fig.2.10 Basis function image of SDHT



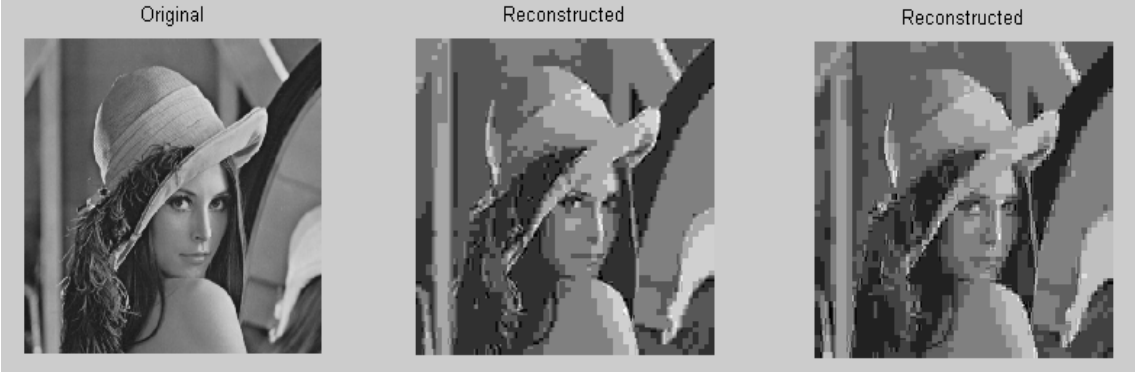
(a)



(b)

Fig.2.11 PSNR performance of SDHT and DCT for (a) Lena image and (b) Cameraman image

both types of images. Therefore, SDHT can be used for the image compression in applications requiring high compression and low hardware cost (low hardware also leads



(a)

(b)

Fig. 2.12 Original (left), reconstructed image using DCT (middle) and reconstructed image using SDHT (right) at very high compressions (a) Lena and (b) Cameraman

to low power consumptions) Fig.2.12 shows the decompressed images at very high compression and TABLE 2.2 shows their performance values in terms of PSNR and compression ratio.

TABLE 2.2
PSNR AND COMPRESSION RATIOS OF IMAGES SHOWN IN FIG.2.12

	Lena Image		Cameraman Image	
Transform used	DCT	SDHT	DCT	SDHT
PSNR	29.44	30.83	30.53	30.95
Compression Ratio	60.88	60.54	48.71	48.93

2.9 Conclusions

Digital image representation and its quality measurement metric have been described in this chapter. The energy compaction property of DCT has been studied by decompressing the standard images with selected low frequency DCT coefficients. High compression can be obtained when images are compressed with few lower DCT coefficients without visual distortion. An alternate transform namely separable discrete Hartley transform (SDHT) has been used for the image compression and decompression in JPEG compression procedure replacing DCT. From simulation, it is found that it performs better than DCT in high compression.

Chapter 3

Distributed Arithmetic and Its VLSI Architecture

3.1 Introduction

Multimedia digital signal processing became more reliable, faster, flexible and cost-effective because of advancement in the VLSI technology. Technology scaling has enabled us to integrate a number of different components on a single VLSI chip. A signal processing algorithm can have different VLSI architectures depending upon the area in which it has to be used. In the early design stage, 'Architecture Exploration' is used to search for the best architecture that meets the desired specifications at the lowest possible component used. DCT is a computation intensive algorithm and is realized by a large number of additions and multiplications operations. Systolic architecture was primarily used for the DSP algorithm implementation because of its modularity and parallel processing. Many DCT architectures were proposed on systolic design to reduce the number of multipliers in the systolic design as multipliers consumes high power and occupy less area. Nevertheless, they could not eliminate it. One solution to complete removal of multipliers from the DSP architecture implementation became possible by the use of distributed arithmetic (DA).

In this chapter, basics of two types of DA architectures, one using ROM and another free of ROM are described. Architectures of 8-point 1-D DHT, 8x8 SDHT, 8-point DCT, 8x8 2-D DCT are described and implemented in both ROM based DA and ROM free DA approach. Comparisons between them are done in terms of hardware and power requirements. An area and power efficient DCT is also proposed and implemented in FPGA as well as standard cell based ASIC.

3.2 Systolic Architecture

Systolic architecture is used to implement digital signal processing and arithmetic algorithms in which fast processing is required. It was the preferred design approach for the special purpose systems because of its simple, regular and pipeline operations done by set of small interconnected similar array cells called processing elements (PEs). Breaking the whole processing into small cells has the two major advantages. First one is that design cost (nonrecurring) is reduced as designing of a small cell cost less as compared to complex one. These small cells are reused for implementing the desired algorithm (Reusability). Second advantage of the systolic architecture is that it is able to

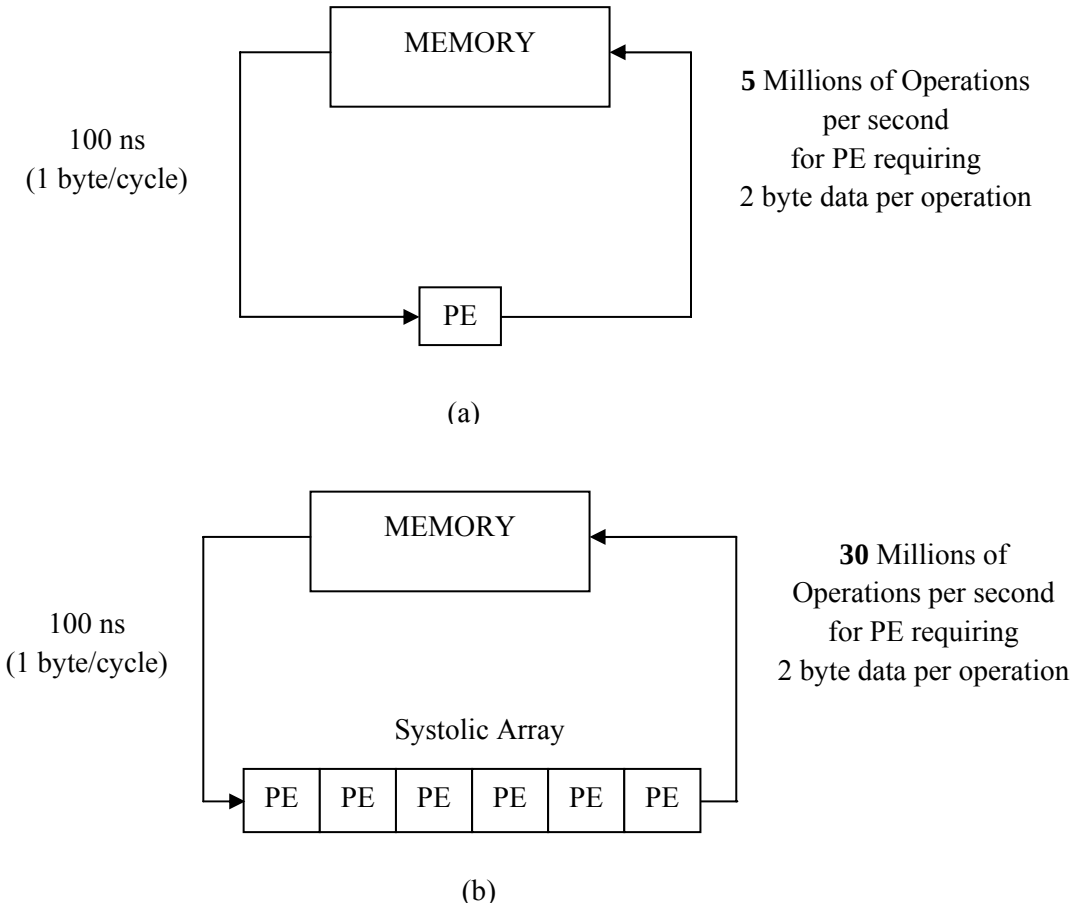


Fig. 3.1 Operations using (a) single processing element and (b) Systolic Array [42]

match the high processing power of its internal components with the slow speed I/O interconnects and lower memory bandwidth. If the I/O speed is low as compared to speed of data being computed by an internal architecture, then high memory bandwidth is required to speed up the computation. In case of systolic architecture, a set of data brought from the memory is processed by several PEs in a pipeline structure i.e., multiple operations are performed on each data item. Therefore, computation is increased at the same memory bandwidth [42]. Fig. 3.1 depicts an example illustrating the increased speed at the same memory bandwidth. Here, the maximum speed between memory processing element is 10 million bytes per second. If the each operation requires two bytes of data, then maximum speed is limited to 5 million operations per second. By the use of multiple processing, the speed is increased to 30 million operations per second at the same bandwidth.

One example for systolic array is given by Kung [42], where the following convolution operation has to be mapped on systolic array. Given a sequence of weights $[w_1, w_2,..,w_k]$ and the input sequence $[x_1,x_2,..,x_n]$, the following result sequence $[y_1, y_2,\ldots,y_{n+1-k}]$ can be obtained which is defined by,

$$y_i = w_1 x_i + w_2 x_{i+1} + \ldots + w_k x_{i+k-1} \tag{3.1}$$

Fig. 3.2 (a) shows the systolic architecture for the convolution with k=3, whereas Fig. 3.2 (b) shows the basic cell (PE) used in the architecture. Multiple data sets have to be convolved with the fixed value w_1,w_2 and w_3. Therefore, to speed up of the operations, each data is available to multiple processing elements and each one perform parallel multiply operations, resulting w_1x_1, w_2x_1 and w_3x_1 in first clock cycle. With the initial value of y set to zero, accumulated result w_1x_1 is pushed to the right cell. In the second clock cycle, this result is accumulated with w_2x_2. After third cycle, final y_i results are available at each clock cycle. A major area and power consuming module in VLSI is multiplier. Past research in the implementation of any DSP algorithm using systolic array was centered on the reduction of the number of multipliers. In many literatures, DCT is implemented by systolic array architectures with the purpose of reducing the number of

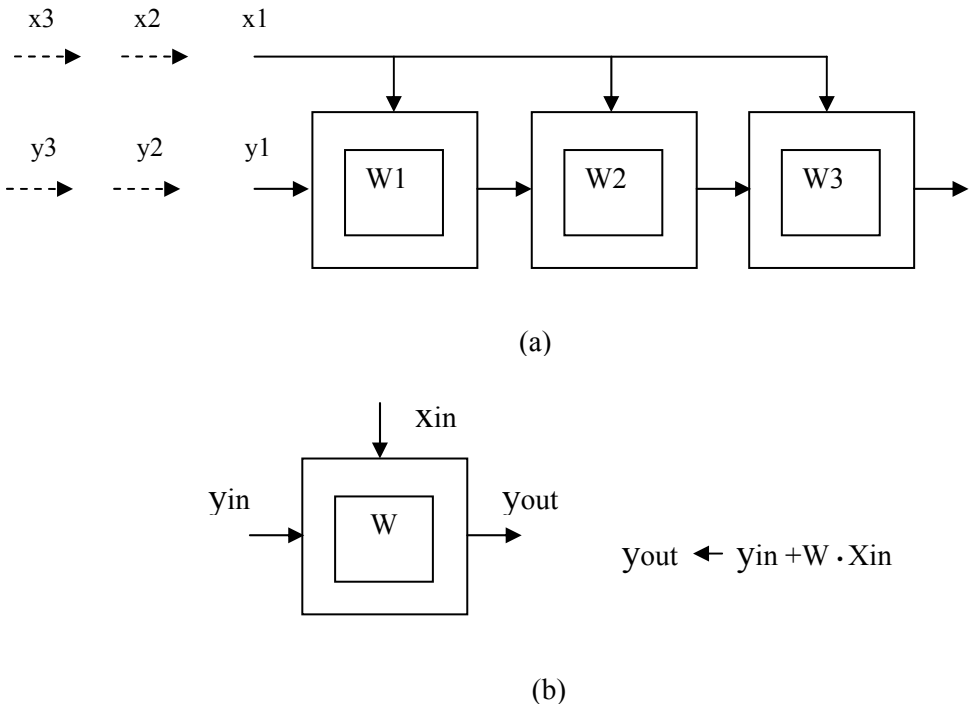


Fig. 3.2 (a) Systolic convolution array and (b) basic operations of one PE [42]

multipliers [43–48]. Apart from systolic, other DCT implementations also favored the multiplier reduction. In the literature by Malvar [49], DCT is implemented using DHT with reduced number of multipliers. Chin *et al.* [50] proposed new convolution based algorithm for computing DCT. Still, all these algorithms are not free of multipliers, i.e. they need multipliers along with adders and other logic components for the VLSI implementation. Next section describes distributed arithmetic (DA) algorithms which implements DSP algorithms without multipliers, which is important for area and power savings in VLSI designs.

3.3 ROM based Distributed Arithmetic (DA)

Distributed Arithmetic (DA) is a bit serial computation approach. It implements multiplications without multiplier in VLSI design. For DA implementation of inner product of arrays, one of the inputs should be constant array. ROM based DA relies on the manual pre-computations of constants and storing them in ROM. These pre-computed

values are fetched from the ROM addressed by the input bits. According to White [51], by careful design, one may reduce the total gate count in a signal processing arithmetic unit upto 80 percent. An example of DA approach for the inner product implementation in signal processing given by White [51] is as follows. Consider the sum of product expressed by,

$$y = \sum_{k=1}^{k} A_k x_k \tag{3.2a}$$

Here, A_k are constant and x_k are other set of inputs. Assuming x_k to be a fractional value (or normalized to fraction), i.e., $|x_k| < 1$, it can be expressed in 2's complement binary representation as,

$$x_k = -b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \tag{3.2b}$$

The b_{k0} is sign bit whereas, $b_{k,N-1}$ is the least significant bit (LSB) in the binary representation of bit-width N. Putting x_k from (3.2b) in (3.2a), y can be written as,

$$\begin{aligned} y &= \sum_{k=1}^{k} A_k \left(-b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \right) \\ &= \sum_{n=1}^{N-1} \left(\sum_{k=1}^{k} A_k \right) b_{kn} 2^{-n} + \sum_{k=1}^{k} A_k (-b_{k0}) \end{aligned} \tag{3.2c}$$

Since A_k are constants, from (3.2c), it can be found that for given number of elements in the inputs (known value of k), expression in the bracket can be pre-computed and stored in ROM. Since, b_{kn} are binary 1 or 0, when multiplied with A_k, it will make A_k either 0 or no change in it. For example, suppose number of inputs are $k = 3$ and number of bits in the binary representation are 4 (N=4), then for a set of inputs [x_1=1010, x_2=1100, x_3= 1001] in 2's complement (first bit being a sign bit), the term in (3.2c) given by,

$$\sum_{n=1}^{N-1} \left(\sum_{k=1}^{k} A_k \right) \times b_{nk}$$

can be written as,

$$\left(\sum_{k=1}^{k} A_k \times b_{1k}\right)+\left(\sum_{k=1}^{k} A_k \times b_{2k}\right)+\left(\sum_{k=1}^{k} A_k \times b_{3k}\right)$$
$$=\left(A_1\times(0)+A_2\times(1)+A_3\times(0)\right)+\left(A_1\times(1)+A_2\times(0)+A_3\times(0)\right) \quad (3.2d)$$
$$+\left(A_1\times(0)+A_2\times(0)+A_3\times(1)\right)$$

The b_{1k} represents bit first of the kth input (which is 0 for first including 1 for second input and 0 for third input) and likewise. Hence, from (3.2d), different combinations of A_k can be pre-computed and stored in memory. The addresses of the pre-computed values will be given by the nth bit of all inputs. In the above example, it will be 010, 100 and 001. The size of the memory (ROM) will be the possible binary combinations of A_k i.e., 2^k and the number of clock cycles required for getting final sum of product will be the number of bits in the input representation. The second term in (3.2c) i.e., $\sum_{k=1}^{k} A_k(-b_{k0})$ has negative sign which means there will be negative storage for all positives as well. So the total memory requirement will be $2x2^k$. TABLE 3.1 shows the binary input combinations. The multiplication with 2^{-n} in the first term will be performed

TABLE 3.1
ROM CONTENTS FOR THREE 4-BITS INPUTS

Inputs (x_k)	b_{1n} (sign-bit)	b_{2n}	b_{3n}	ROM contents
	for $1 \le n \le N-1$			
	0	0	0	$A_1\times0+A_2\times0+A_3\times0=0$
	0	0	1	$A_1\times0+A_2\times0+A_3\times1=A_3$
	0	1	0	$A_1\times0+A_2\times1+A_3\times0=A_2$
	0	1	1	$A_1\times0+A_2\times1+A_3\times1=A_2+A_3$
	1	0	0	$A_1\times1+A_2\times0+A_3\times0=A_1$
	1	0	1	$A_1\times1+A_2\times0+A_3\times1=A_1+A_3$
	1	1	0	$A_1\times1+A_2\times1+A_3\times0=A_1+A_2$
	1	1	1	$A_1\times1+A_2\times1+A_3\times1=A_1+A_2+A_3$
	for n=0			
	0	0	0	$-(A_1\times0+A_2\times0+A_3\times0)=0$
	0	0	1	$-(A_1\times0+A_2\times0+A_3\times1)=-A_3$
	0	1	0	$-(A_1\times0+A_2\times1+A_3\times0)=-A_2$
	0	1	1	$-(A_1\times0+A_2\times1+A_3\times1)=-(A_2+A_3)$
	1	0	0	$-(A_1\times1+A_2\times0+A_3\times0)=-A_1$
	1	0	1	$-(A_1\times1+A_2\times0+A_3\times1)=-(A_1+A_3)$
	1	1	0	$-(A_1\times1+A_2\times1+A_3\times0)=-(A_1+A_2)$
	1	1	1	$-(A_1\times1+A_2\times1+A_3\times1)=-(A_1+A_2+A_3)$

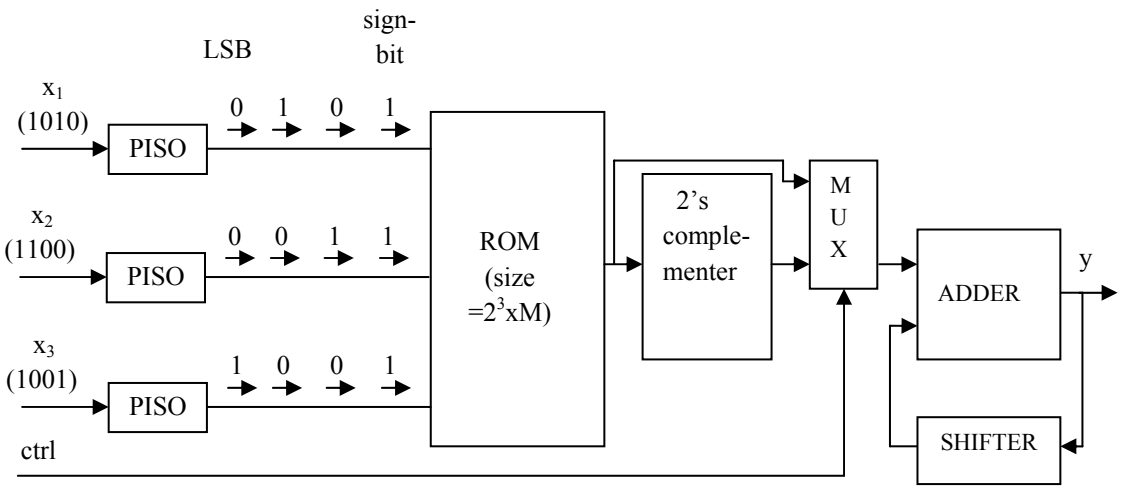


Fig. 3.3 Architecture of ROM based DA

by n-bit right shift operation. Fig. 3.3 depicts the architecture of this ROM based DA. Three inputs (x_1, x_2, x_3, 4-bits each) are applied serially (one bit at each clock cycle, sign-bit first and LSB last) by using parallel to serial converter (PISO) at each input address line of ROM in parallel fashion. Since contents of ROM in the lower half (for n=0) of the TABLE 3.1 is same as upper half (for $1 \leq n \leq N-1$) with a sign change, only one half of the contents need to be stored. By using a control signal (ctrl), sign change can be done by 2's complementer circuit at appropriate time, i.e., at first clock cycle ROM content will be negated. Here, width of the memory is denoted as M and it will determine the precision of the fractional data. Adder will accumulate the data coming from the ROM. Each data from ROM needs to be shifted left by an increasing number of bits every clock cycle before accumulating in adder (first clock cycle 2^{-0}=1, no shift, second clock cycle 2^{-1}=1/2, one bit shift, third clock cycle 2^{-2}=1/4, two bit shift and so on). This can be done by shifting the accumulated result itself one bit at each clock cycle right (left shift of one operand is equivalent to right shift of another). Final result will be obtained in y at N clock cycles (4 clock cycles in the present example).

Offset Binary Coding (OBC) Technique

White [51] has suggested a technique called offset binary coding (OBC) to reduce the ROM size by half. By using the OBC technique, size of ROM can be made half in ROM

based DA. Here are the illustrations for this technique. For interpretation, let input data are not in (0, 1) straight binary code, but in (1, –1) offset binary code. Let,

$$x_k = \frac{1}{2}\left[x_k - (-x_k)\right] \tag{3.3a}$$

Since, x_k is a 2's complement binary, its negative will be expressed as,

$$-x_k = -\bar{b}_{k0} + \sum_{n=1}^{N-1} \bar{b}_{kn} 2^{-n} + 2^{-(N-1)} \tag{3.3b}$$

Here, bar indicates the complement of that bit. From (3.2b) and (3.3b), equation (3.3a) can be written as,

$$x_k = \frac{1}{2}\left[-(b_{k0} - \bar{b}_{k0}) + \sum_{n=1}^{N-1} (b_{kn} - \bar{b}_{kn}) 2^{-n} - 2^{-(N-1)}\right] \tag{3.3c}$$

If we define two new variables c_{kn} and c_{k0} such that $c_{kn} = (b_{kn} - \bar{b}_{kn})$ for $n \neq 0$ and $c_{k0} = -(b_{k0} - \bar{b}_{k0})$, then (3.3c) can be written as,

$$\begin{aligned} x_k &= \frac{1}{2}\left[c_{k0} + \sum_{n=1}^{N-1} c_{kn} 2^{-n} - 2^{-(N-1)}\right] \\ &= \frac{1}{2}\left[\sum_{n=0}^{N-1} c_{kn} 2^{-n} - 2^{-(N-1)}\right] \end{aligned} \tag{3.3d}$$

It is clear that since c_{kn} and c_{k0} are difference of 0 and 1, their values will be either 1 or –1 (0–1=–1 and 1–0=1). So sum of product from equations (3.2a) and (3.3d) are,

$$\begin{aligned} y &= \sum_{k=1}^{k} A_k x_k \\ &= \sum_{k=1}^{k} A_k \left(\frac{1}{2}\left[\sum_{n=0}^{N-1} c_{kn} 2^{-n} - 2^{-(N-1)}\right]\right) \\ &= \frac{1}{2}\sum_{n=0}^{N-1}\left(\sum_{k=1}^{k} A_k c_{kn}\right) 2^{-n} - \frac{1}{2}\sum_{k=1}^{k} A_k 2^{-(N-1)} \end{aligned} \tag{3.3e}$$

Once again, all possible summations of A_k can be pre-computed and stored in ROM.

All possible combinations of A_k summations for k=3 (3 inputs, as mentioned in previous example) are shown in TABLE 3.2. Top half contents of ROM are same as other half with a negative sign. Therefore, only half of 2^k ROM is required and can be addressed by only k–1 bits. In the TABLE 3.2, if first half are stored in ROM, then values corresponding to lower half bits are obtained by inverting bits of the upper half. For example, data at location addressed by [1 0 0] is negative of data addressed by [0 1 1], data addressed by [1 0 1] is negative of data addressed by [0 1 0] and likewise. This can be realized in hardware by doing XOR with bits of first input data. The second expression in equation (3.3e) is a constant which can be stored in a register (k_I) and added with initial condition. Fig 3.4 shows the architecture of ROM based DA using OBC technique.

TABLE 3.2
ROM CONTENTS FOR THREE 4-BITS INPUTS IN OBC TECHNIQUE

	b_{1n} (sign-bit)	b_{2n}	b_{3n}	ROM contents
Inputs (x_k)	0	0	0	$-A_1-A_2-A_3$
	0	0	1	$-A_1-A_2+A_3$
	0	1	0	$-A_1+A_2-A_3$
	0	1	1	$-A_1+A_2+A_3$
	1	0	0	$A_1-A_2-A_3=-(-A_1+A_2+A_3)$
	1	0	1	$A_1-A_2+A_3=-(-A_1+A_2-A_3)$
	1	1	0	$A_1+A_2-A_3=-(-A_1-A_2+A_3)$
	1	1	1	$A_1+A_2+A_3=-(-A_1-A_2-A_3)$

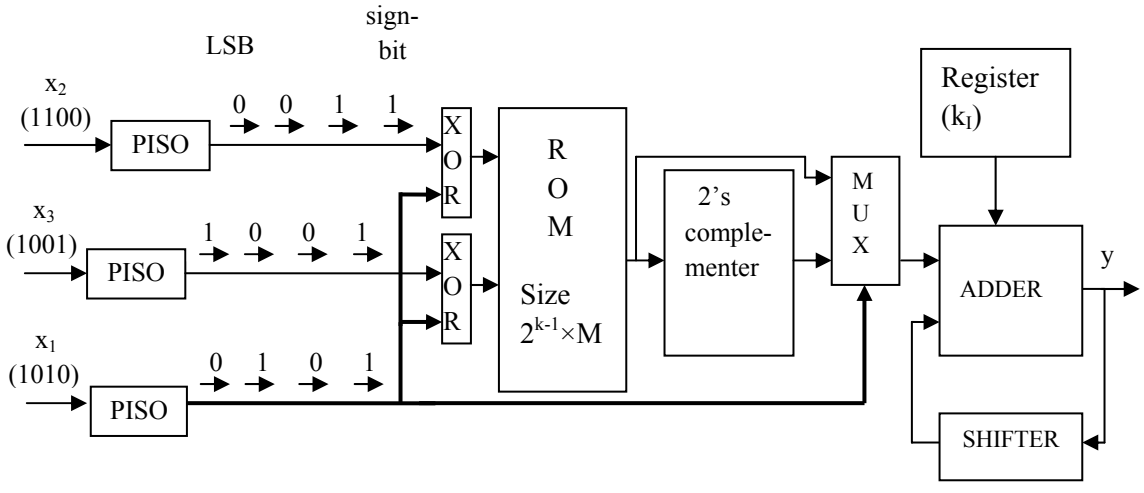


Fig. 3.4 Architecture of ROM based DA using OBC technique

3.3.1 FPGA Implementation of 8-Points 1-D DHT using ROM based DA

1-D DHT implementation using ROM based DA requires less hardware as compared to systolic array implementation [95]. From (2.11), 8-point DHT can be written as,

$$Y(k) = \sum_{n=0}^{7} X(n) cas\left(\frac{2\pi}{8} nk\right), \quad k = 0,1,...,7$$
$$= \sum_{n=0}^{7} X(n) H_{nk}, \quad k = 0,1,...,7 \tag{3.4}$$

For the computation of first coefficient y(1), there will be 8-inputs array multiplication given by,

$$Y(1) = \sum_{n=0}^{7} X(n) H_{n1}$$

Similarly for the other DHT coefficients, 8-array multiplications are required to be done. Since each 8-array multiplication takes ROM of size 2^7 i.e., 128 locations, total number of ROM locations required for 8-point DHT are 8x128. H_{nk} values are pre-computed according to TABLE 3.2 for each coefficient and stored in ROM. We have taken 4-bits precision of fractional binary value stored in ROM. 8-bits inputs and 11-bits output has been taken for the implementation. VHDL code has been written for the 8-point DHT implementation in xc2vp30 device on Virtex-II board of Xilinx FPGA [96]. TABLE 3.3 shows the hardware utilizations of FPGA and Fig. 3.5 depicts the RTL schematic generated. Power analysis is performed using xpower analyzer tool in Xilinx ISE 10.1.

TABLE 3.3
DEVICE UTILIZATION FOR THE FPGA IMPLEMENTATION OF 8-POINT DHT USING ROM BASED DA

FPGA-chip: Xilinx XC2VP30			
	Used	Available	Utilization
# of slices	561	13696	4 %
# of 4 input LUTs	998	27392	3 %
# of slice Flip Flops	341	27392	1 %
Min. Period (ns)	5.92	-	-
Power (mW)	48.8	-	-

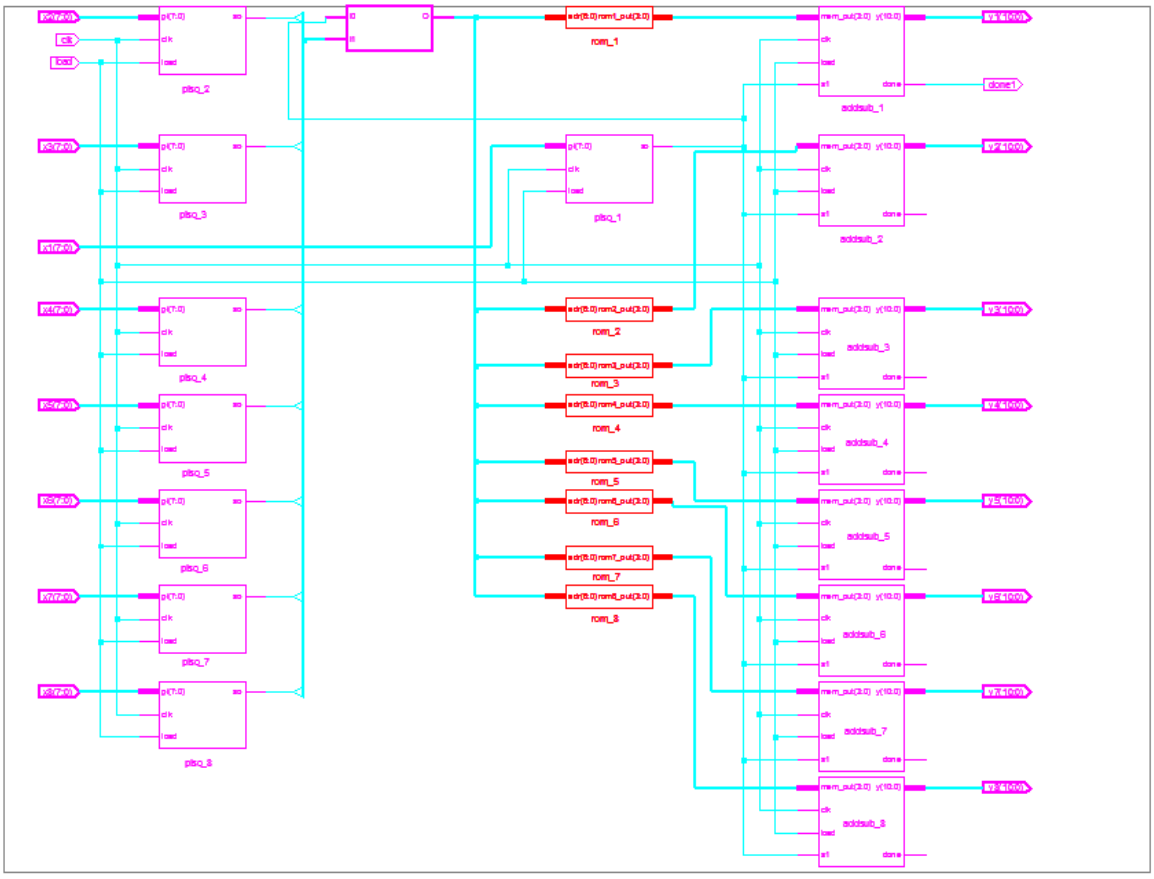


Fig. 3.5 RTL schematic of 8-points DHT using ROM based DA

3.3.2 FPGA Implementation of SDHT using ROM based DA

8x8 2-D SDHT has been implemented using the row-column decomposition technique. First 8-point 1-D DHT is taken to all rows one by one. The transformed 64 1-D DHT coefficients are stored in 64 11-bits width registers. Finally, 8-point 1-D DHT is taken column-wise to 1-D DHT coefficients. Fig. 3.6. depicts the flow.

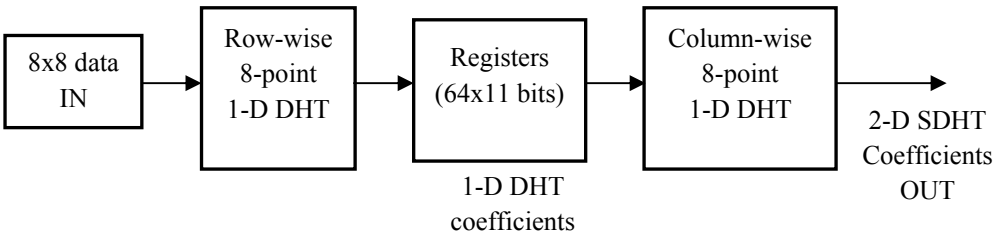


Fig. 3.6 Row-column decomposition technique for 2-D SDHT implementation

A sample 8x8 data is taken for the simulation and hardware verification given by matrix,

$$D_i = \begin{bmatrix} 30 & 29 & 39 & 42 & 32 & 36 & 46 & 39 \\ 33 & 34 & 37 & 36 & 36 & 42 & 43 & 33 \\ 37 & 40 & 34 & 32 & 41 & 45 & 38 & 32 \\ 40 & 43 & 35 & 36 & 45 & 40 & 35 & 47 \\ 40 & 42 & 38 & 42 & 46 & 35 & 43 & 75 \\ 40 & 40 & 41 & 43 & 41 & 40 & 65 & 102 \\ 40 & 40 & 41 & 36 & 35 & 58 & 95 & 117 \\ 41 & 41 & 40 & 27 & 30 & 75 & 117 & 121 \end{bmatrix}$$

and its MATLAB simulation result for 2-D SDHT is given by,

$$D_o(SDHT) = \begin{bmatrix} 2934 & -269 & -360 & -212 & -146 & -94 & 0 & 556 \\ -397 & 48 & 260 & 208 & 128 & 83 & -143 & -370 \\ -214 & 139 & 24 & 29 & 26 & 8 & 120 & -253 \\ -147 & 83 & 41 & 36 & 22 & 11 & 55 & -170 \\ -104 & 50 & 34 & 25 & 16 & 5 & 34 & -117 \\ -64 & 35 & 17 & 17 & 9 & 6 & 21 & -66 \\ 0 & 0 & 6 & 5 & 0 & 0 & -2 & 0 \\ 336 & -238 & -335 & -33 & -48 & -13 & -141 & 426 \end{bmatrix}$$

In Xilinx ISE environment, ChipScope Pro is a tool that integrate the logic analyzer and other measurement hardware components with the target design inside the Xilinx FPGA. Apart, ChipScope Pro contains many features that a designer needs to verify his design. Hardware triggering can be done by external switch without affecting the original design inside the FPGA [97]. Xilinx xc2vp30 device on Virtex-II pro board is programmed by bit-file generated from the ISE 10.1 tool and hardware output is obtained from device using ChipScope Pro logic analyzer tool through USB cable. Fig. 3.7 shows the 2-D SDHT results of 8x8 data matrix D_i. When compared to MATLAB simulations results in D_o(SDHT) matrix, error is present because of fixed point binary representation of fractional data in ROM.

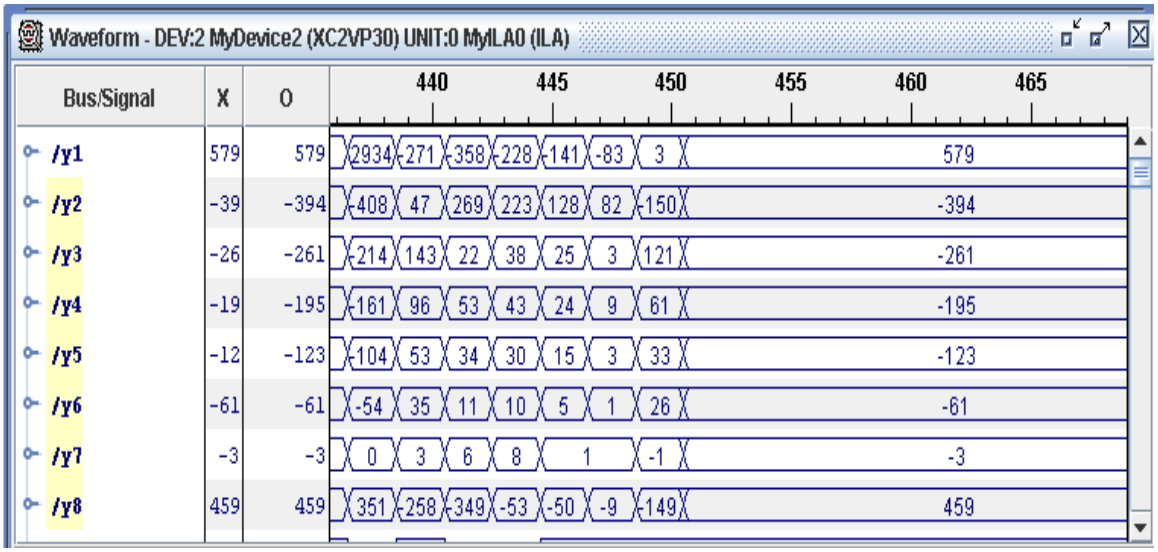


Fig. 3.7 Hardware implementation in Xilinx FPGA of 8x8 data matrix D_i through ChipScope Pro tool

3.4 ROM Free DA

The following are the disadvantages of ROM based DA.

1. Size of ROM is given by $2^{(k-1)}\times M$ where k is size of input M is precision of fractional constant (ROM size increases as the transform size).
2. Bit serial operation along with ROM access makes the overall computation slower (lower throughput).

Also, ROM based design is not preferred choice in VLSI because ROM has slow speed (ROM access time) and more power consumption [55–56]. ROM free DA architecture exploits the sparse nature of matrix formed by binary representation of coefficients (most places are zero) in contrast to pre-computing and storing them in ROM based DA i.e., constant coefficients are distributed. Shams *et. al*. [57] first gave the analysis of ROM free DCT and named the algorithm NEDA (New DA). The following is the description of NEDA.

Consider the sum of product of two vectors given by,

$$y = \sum_{k=1}^{L} A_k X_k \tag{3.5a}$$

which can be written as,

$$y = \begin{bmatrix} A_1 & A_2 & \cdots & A_L \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_L \end{bmatrix} \tag{3.5b}$$

where, A_k are the constant coefficients and X_k are the input variables. In binary 2's complement form A_k can be expressed as,

$$A_k = -A_{k,N-1}2^{N-1} + \sum_{n=0}^{N-2} A_{k,n}2^n \tag{3.5c}$$

$A_{k,n}$ are bit 0 or 1. N is the number of bits in binary representation of A_k and $A_{k,N-1}$ represents the sign-bit while $A_{k,0}$ the LSB. Here, N is referred as DA precision. Equation (3.5c) can be expressed in matrix product form given by,

$$A_k = \begin{bmatrix} 2^0 & 2^1 & \cdots & 2^{N-1} \end{bmatrix} \cdot \begin{bmatrix} A_{k,0} \\ A_{k,1} \\ \vdots \\ -A_{k,N-1} \end{bmatrix} \tag{3.5d}$$

From (3.5d), for each k, equation (3.5b) can be written as,

$$y = \begin{bmatrix} 2^0 & 2^1 & \cdots & 2^{N-1} \end{bmatrix} \cdot \begin{bmatrix} A_{1,0} & A_{2,0} & \cdots & A_{L,0} \\ A_{1,1} & A_{2,1} & \cdots & A_{L,1} \\ \vdots & \vdots & \cdots & \vdots \\ -A_{1,N-1} & -A_{2,N-1} & \cdots & -A_{L,N-1} \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_L \end{bmatrix}$$

$$y = \begin{bmatrix} 2^0 & 2^1 & \cdots & 2^{N-1} \end{bmatrix} \cdot M \cdot \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_L \end{bmatrix} \tag{3.5e}$$

Here, matrix M is a sparse matrix as it contains the binary values either 0 or 1. From equation (3.5e), it is evident that summation y can be realized by using adders only and

also less number of adders will be used depending upon the number of 1's present in the sparse matrix *M*. For example, suppose two sets of vectors are given by A_1, A_2, A_3 as constant coefficients and X_1, X_2, X_3 as variables. If binary 2's complement representation of *A* are,

$$A_1 = [1 \quad 0 \quad 0 \quad 1], \quad A_2 = [0 \quad 1 \quad 0 \quad 1], \quad A_3 = [1 \quad 1 \quad 0 \quad 1]$$

then from equation (3.5e), their sum of product is,

$$y = \begin{bmatrix} 2^0 & 2^1 & 2^2 & 2^3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \\ -1 & -0 & -1 \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix}$$

$$= \begin{bmatrix} 2^0 & 2^1 & 2^2 & 2^3 \end{bmatrix} \cdot \begin{bmatrix} X_1 + X_2 + X_3 \\ 0 \\ X_2 + X_3 \\ -X_1 - X_3 \end{bmatrix}$$

$$= (X_1 + X_2 + X_3) \cdot 2^0 + (X_2 + X_3) \cdot 2^2 - (X_1 + X_3) \cdot 2^3 \tag{3.5f}$$

Expression in (3.5f) can be realized by the hardware architecture shown in Fig. 3.8 where, '+' and '–' signs represent adder and subtracter respectively. The structure is free

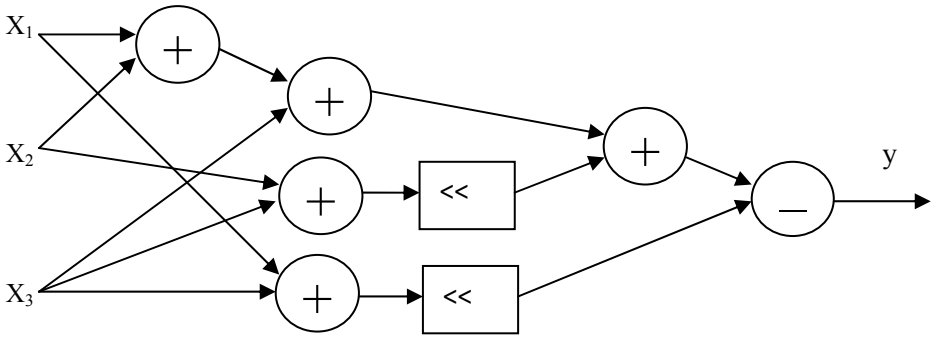


Fig. 3.8 Structure to realize the sum of vectors in the example using ROM free DA

of ROM and multipliers. It can be implemented in a single clock cycle with the help of adders/subtracters and shifters only. Therefore, ROM free DA is faster. Further, adders/subtracters can be compressed in number of bits to reduce the area of design. For example, shifting the binary values right results in less number of bits in its representation and hence, bit width of the adders can be reduced accordingly. Bit width reduction of adders in ROM based DA leads to low accuracy. Following are the differences between ROM based DA and ROM free DA for sum of product implementation.

1. Number of adders in ROM free DA is proportional to number of 1's present in binary representation of constant coefficients but in case of ROM based DA, number of adders and shifters requirement is 1 each along with ROM of size 2^{k-1}.
2. With the increased number of bits in input representation, width of adders in ROM free DA will increase but the number remains the same whereas, in ROM based DA, it will lead to increased clock cycle and hence slower speed.

3.4.1 FPGA Implementation of DCT using ROM free DA

Chungan et al. [58] has implemented 8x8 2-DCT in standard cell technology library using ROM free DA. Adders are claimed to have more compressed in as compared to DCT NEDA architecture of DCT in [57]. For 8-point 1-D DCT, equation (2.9) can be written as,

$$F(u)=\frac{1}{2}C(u)\sum_{i=0}^{7}X(i)\cos\left(\frac{(2i+1)u\pi}{16}\right) \tag{3.6}$$

In [65], (3.6) can be broken using periodicity properties as,

$$F(0)=[X(0)+X(1)+X(2)+X(3)+X(4)+X(5)+X(6)+X(7)]P \tag{3.7a}$$

$$F(1)=[X(0)-X(7)]A+[X(1)-X(6)]B+[X(2)-X(5)]C+[X(3)-X(4)]D \tag{3.7b}$$

$$F(2)=[X(0)-X(3)-X(4)+X(7)]M+[X(1)-X(2)-X(5)+X(6)]N \tag{3.7c}$$

$$F(3)=[X(0)-X(7)]B+[X(1)-X(6)](-D)+[X(2)-X(5)](-A)+[X(3)-X(4)](-C) \tag{3.7d}$$

$$F(4)=[X(0)-X(1)-X(2)+X(3)+X(4)-X(5)-X(6)+X(7)]P \tag{3.7e}$$

$$F(5) = [X(0) - X(7)]C + [X(1) - X(6)](-A) + [X(2) - X(5)]D + [X(3) - X(4)]B \quad (3.7f)$$

$$F(6) = [X(0) - X(3) - X(4) + X(7)]N + [X(1) - X(2) - X(5) + X(6)](-M) \quad (3.7g)$$

$$F(7) = [X(0) - X(7)]D + [X(1) - X(6)](-C) + [X(2) - X(5)]B + [X(3) - X(4)](-A) \quad (3.7h)$$

where,

$$M = \frac{1}{2}\cos\frac{\pi}{8}, \quad N = \frac{1}{2}\cos\frac{3\pi}{8}, \quad P = \frac{1}{2}\cos\frac{\pi}{4},$$

$$A = \frac{1}{2}\cos\frac{\pi}{16}, \quad B = \frac{1}{2}\cos\frac{3\pi}{16}, \quad C = \frac{1}{2}\cos\frac{5\pi}{16}, \quad D = \frac{1}{2}\cos\frac{7\pi}{16}$$

Now, ROM free DA based algorithm can be used to implement DCT equation above. Constant cosine coefficients can be written in 2's complement binary fractional form to exploit the DA. For example, $F(1)$ coefficient can be written with 12-bit DA precision according to (3.5e) as,

$$F(1) = \begin{bmatrix} -2^0 & 2^{-1} & \cdots & 2^{-12} \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{2}\cos\left(\pi/16\right) & \frac{1}{2}\cos\left(3\pi/16\right) & \frac{1}{2}\cos\left(5\pi/16\right) & \frac{1}{2}\cos\left(7\pi/16\right) \end{bmatrix} \cdot \begin{bmatrix} (X(0) - x(7)) \\ (X(1) - x(6)) \\ (X(2) - x(5)) \\ (X(3) - x(6)) \end{bmatrix}$$

$$= \begin{bmatrix} -2^0 & 2^{-1} & \cdots & 2^{-12} \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} (X(0) - x(7)) \\ (X(1) - x(6)) \\ (X(2) - x(5)) \\ (X(3) - x(6)) \end{bmatrix} \quad (3.8)$$

It can be noted that negative sign is used for power of 2 because here binary representation is for fractional value instead of integral value in equation (3.5e) and can be implemented with right shift. If we write $F(1)$ in the form as,

$$F(1) = \begin{bmatrix} -2^0 & 2^{-1} & \cdots & 2^{-12} \end{bmatrix} \begin{bmatrix} F^0(1) \\ F^1(1) \\ F^2(1) \\ F^3(1) \\ F^4(1) \\ F^5(1) \\ F^6(1) \\ F^7(1) \\ F^8(1) \\ F^9(1) \\ F^{10}(1) \\ F^{11}(1) \\ F^{12}(1) \end{bmatrix} \tag{3.9}$$

where powers of F denote number of times shifting is required after evaluating right most

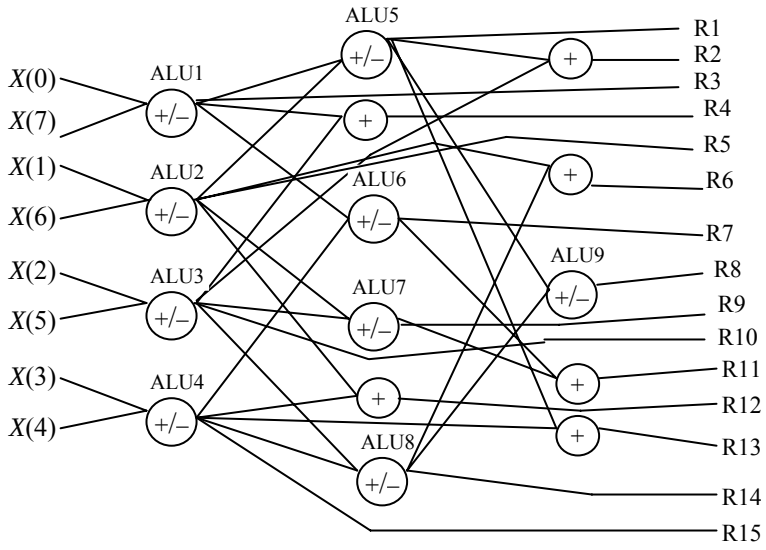


Fig. 3.9 Adder/subtracter structure to realize the 8-points DCT of equation (3.8) [58]

TABLE 3.4

FUNCTIONS OF EACH ALU FOR DIFFERENT DCT COEFFICIENTS [58]

	F(0)	F(1)	F(2)	F(3)	F(4)	F(5)	F(6)	F(7)
ALU1	+	–	+	–	+	–	+	–
ALU2	+	–	+	–	+	–	+	–
ALU3	+	–	+	–	+	–	+	–
ALU4	+	–	+	–	+	–	+	–
ALU5	+	+	+	+	–	+	+	+
ALU6	+	+	–	+	+	+	–	+
ALU7	+	+	–	+	+	+	–	+
ALU8	+	+	+	+	–	+	+	+
ALU9	+	+	+	+	–	+	+	+
F^0	0	0	0	R6	0	R5	R9	R12
F^1	0	0	0	R6	0	R5	R9	R12
F^2	R8	R2	R7	R1	R8	R7	0	R10
F^3	0	R1	R11	R13	0	R15	R7	R1
F^4	R8	R7	R11	R15	R8	R10	R7	R1
F^5	R8	R13	0	R7	R8	R14	R9	R2
F^6	0	R4	R7	R5	0	R3	0	0
F^7	R8	R9	R7	R2	R8	R13	0	R14
F^8	0	R4	0	R5	0	R3	R9	0
F^9	R8	R7	R9	R14	R8	R9	R11	R13
F^{10}	0	R12	R11	R7	0	R14	R11	R2
F^{11}	0	R12	R7	R7	0	R14	R7	R2
F^{12}	0	R14	R7	R12	0	R4	R7	R1

two matrices, then whole 8 DCT coefficients can be calculated by using adder/subtracter structure in Fig. 3.9 and TABLE 3.4 combined [58]. The '+' and '–' sign in the corresponding row indicates function (addition or subtraction) to be performed by the ALUs in Fig. 3.9. Each DCT coefficient is obtained by summing R_i values in the same column. Shifting must be done before summing the column values and number of bit to be shifted right is decided by the power of F in that row. For example, coefficient F(1) is calculated as,

$$F(1) = R2/2^2 + R1/2^3 + R7/2^4 + R13/2^5 + R4/2^6 + R9/2^7 + R4/2^8 + R7/2^9 + R12/2^{10} + R12/2^{11} + R14/2^{12} \quad (3.10)$$

and R_i values from Fig. 3.9 and TABLE 3.4 are,

$$R2 = \big((X(0) - X(7)) + (X(1) - X(6))\big) + (X(2) - X(5)),$$
$$R1 = (X(0) - X(7)) + (X(1) - X(6)),$$
$$R7 = (X(0) - X(7)) + (X(3) - X(4)),$$
$$R13 = \big((X(0) - X(7)) + (X(1) - X(6))\big) + (X(0) - X(7)),$$
$$R4 = (X(0) - X(7)) + (X(2) - X(5)),$$
$$R9 = (X(1) - X(6)) + (X(2) - X(5)),$$
$$R12 = (X(1) - X(6)) + (X(3) - X(4)),$$
$$R14 = (X(2) - X(5)) + (X(3) - X(4))$$

as ALU1, ALU2, ALU3 and ALU4 perform subtraction whereas, others perform addition operation. Division operations of power of 2 in (3.10) is performed by shifting the corresponding R_i values right. Since the binary contents become smaller after shifting i.e., bit width decreases, adders width can be made smaller accordingly to reduce the area. For example, if R_1 and R_2 sizes are of 10-bits each, then adder bit width required to add them should be of size 10-bits. Since R_2 is shifted by 2-bits and R_1 by 3-bits right, it can be realized by using 8-bits adder as shown in Fig. 3.10.

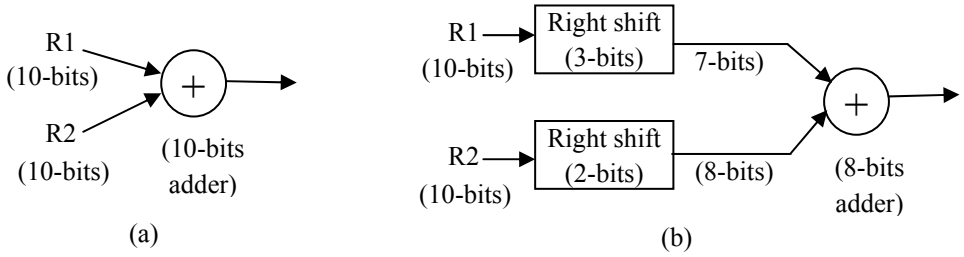


Fig. 3.10 Adder bit width reduction in ROM free DA to save area and power (a) without shift and (b) with right shift

Adder for error reduction in shift and add method

Shifted data are represented by less number of bits and hence, adder bit-width is reduced resulting in less hardware cost, as explained above. For DCT computation, image data is represented in signed 2's complement form range -128 to 127. Bit width of shifted data is determined by number of times shift operation is done. So different bit-width intermediate data are present which are to be added. For 2-input adder, both input data width has to be equal and hence, sign extension is done in smaller bit-width data. Shifting

and addition with sign extension creates error. For example, if initial value is -2 in 8-bit 2's complement representation, this can be written as 11111110. If we take 4 shifted sample of this value a=111111 (shifting 2 times), b=1111 (shifting 4 times), c=111 (shifting 5 times), d=11 (shifting 6 times) all are -1. But all these data should be zero. If we add these values in cascade, result will be -4 (which should be zero). To overcome this problem, we have realized adder as shown in Fig. 3.11. If one of the inputs is -1, then output is other input.

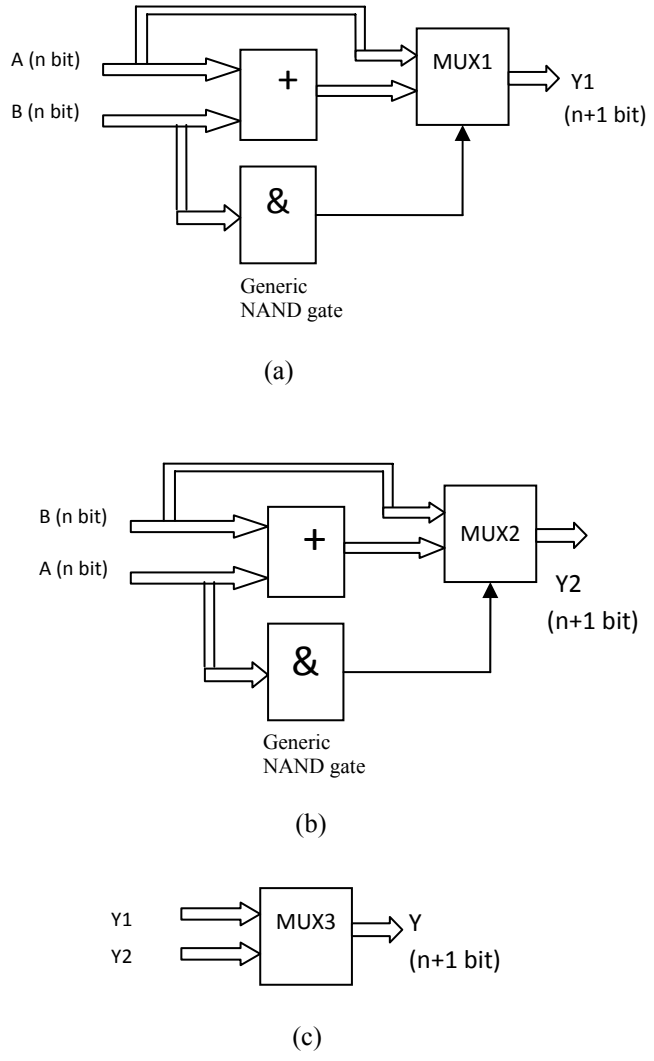


Fig. 3.11 Circuits to reduce sign extension error propagation when number is negative (a) MUX1 selects A if B is -1 else sum of A and B (b) MUX2 selects B if A is -1 else sum of A and B, (c) Final sum is from MUX1 or MUX2 output.

FPGA and ASIC Implementation

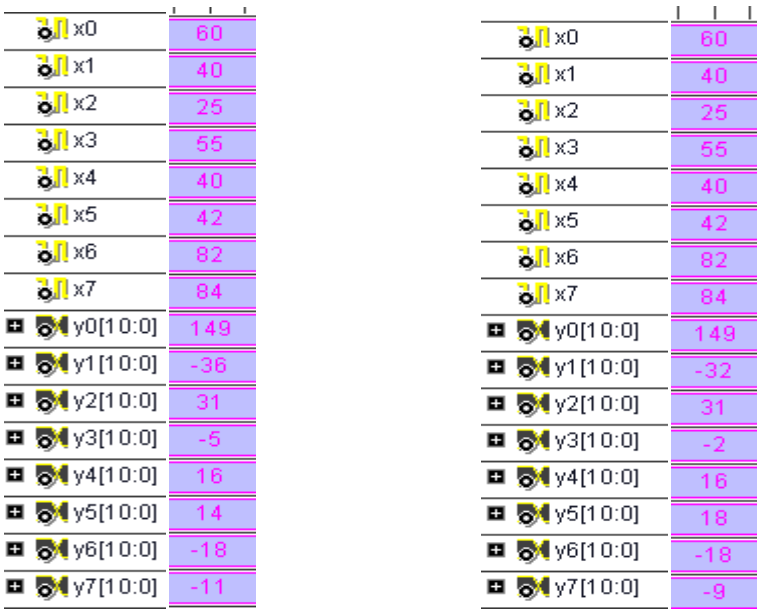
VHDL code has been written for the FPGA as well as ASIC implementations. Simulation is performed using Xilinx ISE simulator for 8x1 data matrix X which is given by,

$$X = [60, 40, 25, 55, 40, 42, 82, 84]$$

MATLAB simulation result for 1-D DCT of X gives,

$$Y = [151.3209 \ \ -32.4895 \ \ 33.1588 \ \ -1.7108 \ \ 17.6777 \ \ 18.5074 \ \ -16.0309 \ \ -5.0975]$$

Fig. 3.12 shows the result of Xilinx ISE simulator using simple adder and proposed adder scheme. It is evident with MATLAB comparison that error due to sign extension is less in proposed adder scheme.



(a) (b)

Fig. 3.12 VHDL simulation result using Xilinx ISE Simulator of data X for the implementation of 1-D DCT architecture using (a) simple addition operator and (b) proposed adder

For FPGA implementation of DCT as discussed above, xc2vp30 device on Xilinx Virtex-II pro board is used. TABLE 3.5 shows the device utilization summary for the FPGA implementation. For the ASIC implementation, TSMC CLN65GPLUS 65 nm

standard cell technology library has been used and the code is synthesized in Synopsys Design Compiler (DC). TABLE 3.6 shows the hardware requirements in terms of area and power 8-bits input data and 12-bits DA precision has been used in both the implementations.

TABLE 3.5
DEVICE UTILIZATION FOR THE FPGA IMPLEMENTATION OF 8-POINT 1-D DCT

FPGA-chip: Xilinx XC2VP30	
# of 4 input LUTs	1268
# of slices	694
# of slice Flip Flops	0
# of IOB Flip Flops	88
Min. Period (ns)	32.6
Power (W)	13.1

TABLE 3.6
AREA AND POWER COMPARISONS FOR SYNOPSYS DC IMPLEMENTATION OF 1-D DCT

TSMC CLN65GPLUS 65nm technology	
Total cell area	8259.84 μm^2
Total Dynamic Power (global operating voltage 1.1v)	3.62 mW
Min. Slack at 500MHz	0.004

3.4.2 Area and Power Efficient VLSI Architecture of 8X1 1-D DCT

By analyzing equations (3.7a) to (3.7h) in previous section, it can be seen that there are only seven cosine terms that are to be represented in DA form. So instead of computing F(0) to F(7) in parallel as in [58], they can be computed in pipelined fashion. A general DA based module can be implemented which takes the inputs and gives the multiply and accumulation result.

Let,

$$a1=X(0)+X(1)+X(2)+X(3)+X(4)+X(5)+X(6)+X(7),$$
$$a2=X(0)-X(1)-X(2)+X(3)+X(4)-X(5)-X(6)+X(7),$$
$$b1=X(0)-X(7),\ b2=X(1)-X(6),\ b3=X(2)-X(5),\ b4=X(3)-X(4),$$
$$c1=X(0)-X(3)-X(4)+X(7) \text{ and } c2=X(1)-X(2)-X(5)+X(6)$$

Then from (3.7a) to (3.7h),

$$\begin{aligned}
&F(0)=(a1\times P),\ F(4)=(a2\times P),\\
&F(1)=(b1\times A)+(b2\times B)+(b3\times C)+(b4\times D),\\
&F(3)=(b1\times B)-(b2\times D)-(b3\times A)-(b4\times C),\\
&F(5)=(b1\times C)-(b2\times A)+(b3\times D)+(b4\times B),\\
&F(7)=(b1\times D)-(b2\times C)+(b3\times B)-(b4\times A),\\
&F(2)=(c1\times M)+(c2\times N),\ \text{and}\ F(6)=(c1\times N)-(c2\times M)
\end{aligned} \tag{3.11}$$

These equations can be implemented using only 4 DA modules as compared to 7 DA modules used in previous implementation. Therefore, area and power reduction can be achieved. These modules will be given by,

$$\begin{bmatrix} -2^{0} & 2^{-1} & \cdots & 2^{-12} \end{bmatrix} \cdot [P] \cdot [a1], \tag{3.12a}$$

$$\begin{bmatrix} -2^{0} & 2^{-1} & \cdots & 2^{-12} \end{bmatrix} \cdot \begin{bmatrix} A & B & C & D \end{bmatrix} \cdot \begin{bmatrix} b1 \\ b2 \\ b3 \\ b4 \end{bmatrix}, \tag{3.12b}$$

and

$$\begin{bmatrix} -2^{0} & 2^{-1} & \cdots & 2^{-12} \end{bmatrix} \cdot \begin{bmatrix} M & N \end{bmatrix} \cdot \begin{bmatrix} c1 \\ c2 \end{bmatrix} \tag{3.12c}$$

F(*0*) and *F*(*4*) coefficients can be obtained from the DA module in (3.12a) in two clock cycles with a1 as input in first clock cycle for coefficient *F*(*0*) and *a2* as input in second clock cycle for *F*(*4*) coefficient. In similar way, from other two modules, rest of the DCT coefficients can be calculated by input ordering and sign change. Fig. 3.13 shows the hardware architectures and TABLE 3.7 shows the ordering of the inputs at each clock cycle. Timing and control unit determines the inputs at each clock cycle and also sign of the inputs. It gives the signal 0 or 1 to multiplexers to select input having same sign or inverted.

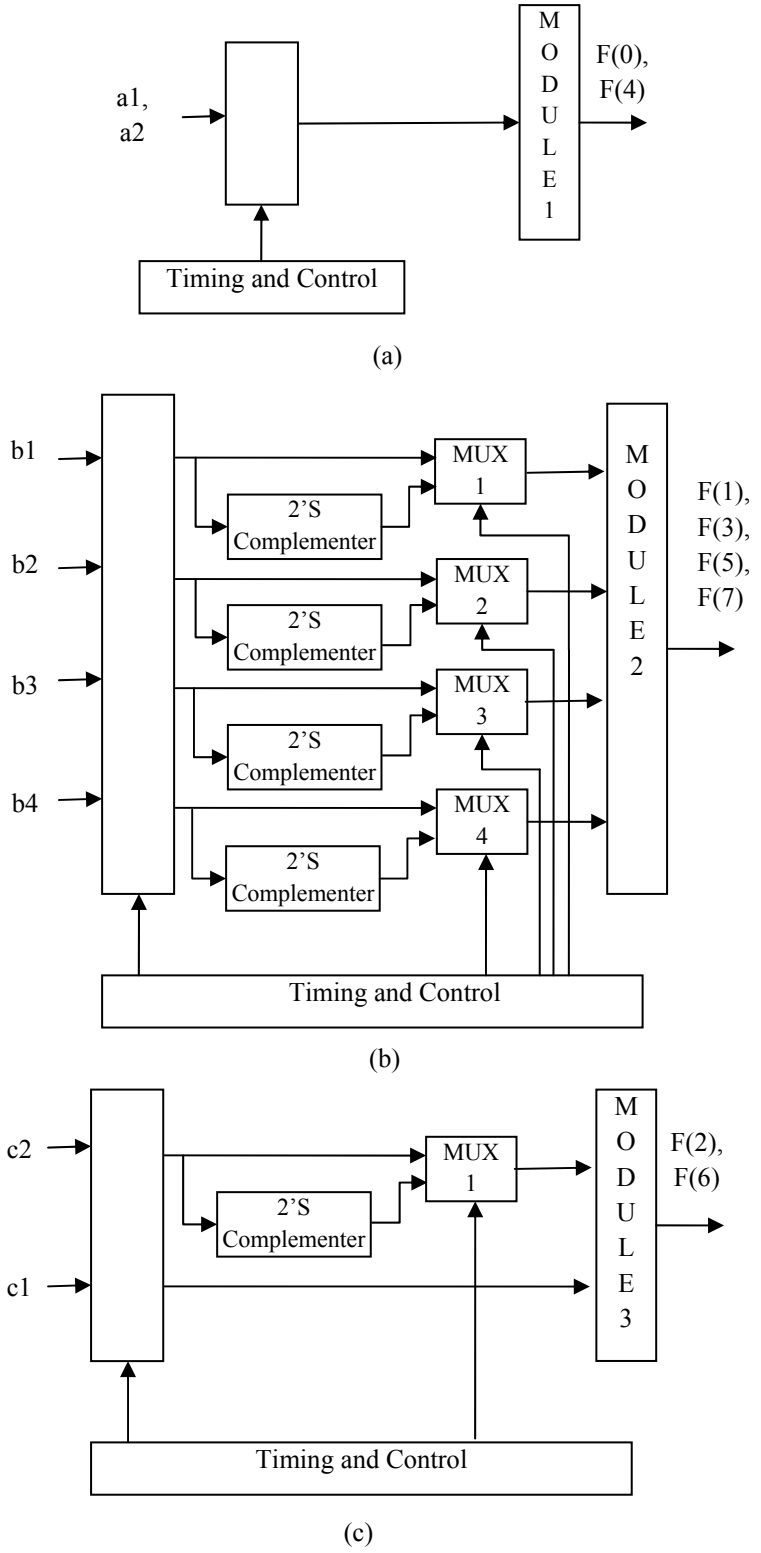


Fig. 3.13 VLSI architecture for computation of 8 point DCT in pipeline manner for (a) computation of F(0) and F(4) (b) computation of F(1), F(3), F(5) and F(7) and (c) computation of F(2) and F(6)

TABLE 3.7(a)
PIPELINE COMPUTATION OF DCT COEFFICIENTS F(0) AND F(4)

MODULE 1		
	Clock cycle 1	Clock cycle 2
Input	a1	a2
Output	F(0)	F(4)

TABLE 3.7(b)
PIPELINE COMPUTATION OF DCT COEFFICIENTS F(1), F(3), F(5) AND F(7)

MODULE 2				
	Clock cycle 1	Clock cycle 2	Clock cycle 3	Clock cycle 4
Input 1	b1	-b3	-b2	-b4
Input 2	b2	b1	b4	b3
Input 3	b3	-b4	b1	-b2
Input 3	b4	-b2	b3	b1
Output	F(1)	F(3)	F(5)	F(7)

TABLE 3.7(c)
PIPELINE COMPUTATION OF DCT COEFFICIENTS F(2) AND F(6)

MODULE 3		
	Clock cycle 1	Clock cycle 2
Input 1	c1	-c2
Input 2	c2	c1
Output	F(2)	F(6)

Hardware Implementation results and comparisons

The architecture proposed (Fig. 3.13) has been implemented in Xilinx FPGA and also in TSMC CLN65GPLUS 65 nm standard cell technology library for ASIC using VHDL code. We have implemented ROM free DA architecture proposed in [58] in FPGA. We have compared the proposed ROM free DA architecture for 8-points 1-D DCT with the ROM free DA architecture in [58]. TABLE 3.8 summarizes the FPGA comparisons while TABLE 3.9 compares the ASIC implementations with 8-bits input and 12-bits DA precision. Fig. 3.14 depicts the RTL schematic in Xilinx ISE 10.1.

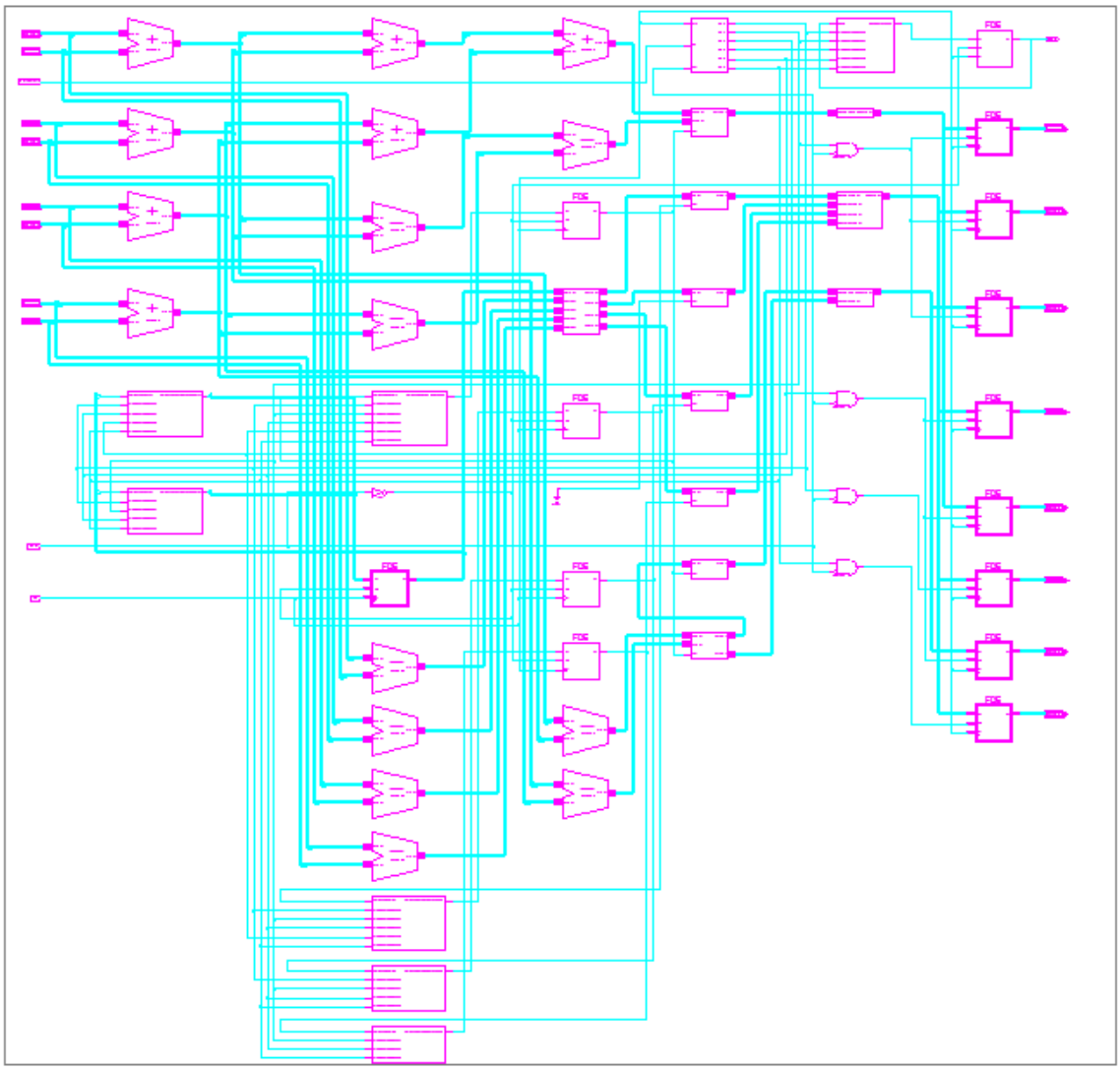


Fig. 3.14 RTL Schematic of Proposed 8-point 1-D DCT in Xilinx ISE 10.1

TABLE 3.8
DEVICE UTILIZATION FOR THE FPGA IMPLEMENTATION OF 8-POINT 1-D DCT

FPGA-chip: Xilinx XC2VP30		
	1-D DCT architecture in [58]	Proposed 1-D DCT architecture
# of 4 input LUTs	1268	696
# of slices	694	370
# of slice Flip Flops	0	97
# of IOB Flip Flops	88	0
Min. Period (ns)	32.6	16.29 (Freq. 61.38 MHz)
Power (W)	13.1	2.06

TABLE 3.9
AREA AND POWER COMPARISONS FOR SYNOPSYS DC IMPLEMENTATION OF 8-POINT 1-D DCT

TSMC CLN65GPLUS 65nm technology			
	1-D DCT architecture in [58]	Proposed 1-D DCT architecture	improvement
Total cell area	8259.84	5683.68	31.2 %
Total Dynamic Power (global operating voltage 1.1v)	3.62 mW	2.27 mW	37.3 %
Min. Slack at 500 MHz	0.004	0.116	

Total of 31.2 % area and 37.3 % power improvements are achieved in standard cell based synthesis of VHDL code. From the FPGA and ASIC implementation comparison results, it is evident that the proposed architecture is efficient in terms of area (FPGA resources in case of FPGA implementation) and power.

Using row-column decomposition technique, 8x8 2-D DCT is implemented using proposed architecture. Intermediate 1-D DCT results are stored in registers. TABLE 3.10 shows the FPGA implementation result and TABLE 3.11 shows the ASIC implementation result.

TABLE 3.10
DEVICE UTILIZATION SUMMARY FOR 2-D DCT IMPLEMENTATION USING ROW-COLUMN DECOMPOSITION TECHNIQUE OF PROPOSED 1-D DCT ARCHITECTURE

FPGA-chip: Xilinx XC2VP30	
# of 4 input LUTs	2522
# of slices	1701
# of slice Flip Flops	1025
Max. Freq.(MHz)	45.173
Power (W)	0.751

TABLE 3.11
2-D DCT ARCHITECTURE IMPLEMENTATION AREA AND POWER USING ROW-COLUMN DECOMPOSITION TECHNIQUE OF PROPOSED 1-D DCT

TSMC CLN65GPLUS 65 nm technology	
Total cell area	23505.84 μm^2
Total Dynamic Power (global operating voltage 1.1v)	5.78 mW
Min. Slack at 500 MHz	0.036

3.5 SDHT Implementation using ROM Free DA

Considering the periodicity and symmetry of trigonometric functions 8-point 1-D DHT equation (3.4) can be written as,

$$y(0) = [x(0) + x(4) + x(1) + x(5) + x(2) + x(6) + x(3) + x(7)]A$$
$$y(1) = [x(1) - x(5)]C + [x(2) - x(6)]B + [x(0) - x(4)]A$$
$$y(2) = [x(2) + x(6)](-A) + [x(1) + x(5)]B + [x(3) + x(7)](-B) + [x(0) + x(4)]A$$
$$y(3) = [x(2) - x(6)](-B) + [x(3) - x(7)]C + [x(0) - x(4)]A$$
$$y(4) = [x(1) + x(5)](-A) + [x(2) + x(6)]A + [x(3) + x(7)](-A) + [x(0) + x(4)]A$$
$$y(5) = [x(1) - x(5)](-C) + [x(2) - x(6)]B + [x(0) - x(4)]A$$
$$y(6) = [x(0) + x(4)]A + [x(1) + x(5)](-B) + [x(2) + x(6)](-A) + [x(3) + x(7)]B$$
$$y(7) = [x(2) - x(6)](-B) + [x(3) - x(7)](-C) + [x(0) - x(4)]A$$

Representing in DA form as in [58] for DCT, we get adder/subtractor matrix for DHT for all data as in Fig.3.15. TABLE 3.12 shows the explanation of Fig.3.15. Divide and multiply operations are done by shifting. Y^n implies shifting n bits. Negative sign in n implies left shift where as positive sign implies right shift. Positive sign in TABLE 3.12 implies that ALUs perform addition and negative sign implies subtraction operation as for DCT in previous section (Section 3.3). As an example, let's take the fourth column for calculating Y(2). Y(2) is the sum of Y-1(2)*2, Y0(2), Y1(2)/2, Y2(2)/22, Y3(2)/23, Y4(2)/24, Y5(2)/25, Y6(2)/26, and Y7(2)/27. The values of Y-1(2), Y0(2), Y1(2), Y2(2), Y3(2), Y4(2), Y5(2), Y6(2), Y7(2) can be obtained from R3, R5, 0, 0, 0, 0, 0, 0, 0 in TABLE 3.12. So, Y(2) can be calculated as,

$$Y(2)=R3*2+R5$$

TABLE 3.13 shows the number of adder/subtracter used in the implementation and bit-width comparison for DCT and DHT. Compared to DCT adder/subtractor of [58], DHT adder/subtractor requires less number of ALU and adders (only 4 ALUs and 7 adders in proposed DHT, but 9 ALUs and 6 adders in DCT).

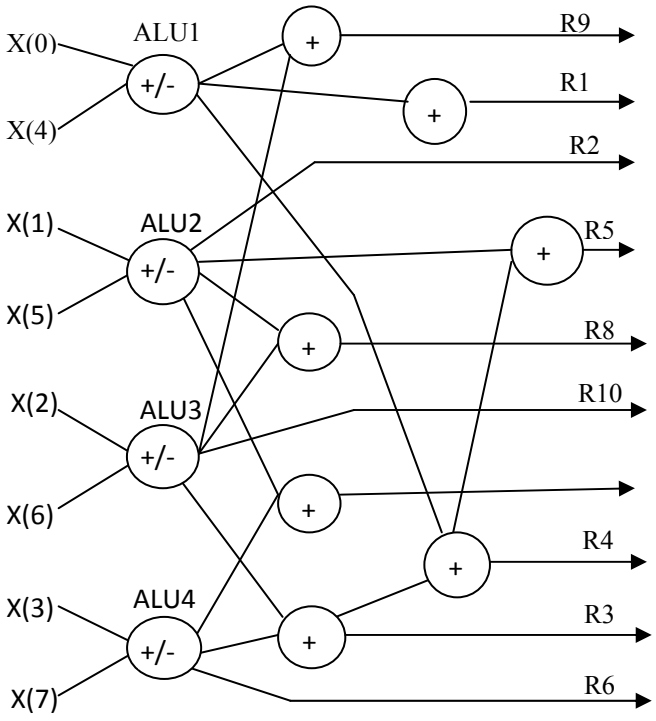


Fig. 3.15 Adder/subtracter for all 8-point DHT coefficients calculation

TABLE 3.12
FUNCTIONS OF EACH ALU FOR DIFFERENT DHT COEFFICIENTS

	Y(0)	Y(1)	Y(2)	Y(3)	Y(4)	Y(5)	Y(6)	Y(7)
ALU1	+	-	+	-	+	-	+	-
ALU2	+	-	+	NO	+	-	+	NO
ALU3	+	-	+	-	+	-	+	-
ALU4	+	NO	+	-	+	NO	+	-
Y^{-1}	0	0	R3	R10	R7	R2	R8	R3
Y^{0}	R5	R1	R5	R4	R5	R9	R5	R9
Y^{1}	0	0	0	0	0	R2	0	R6
Y^{2}	0	R2	0	R6	0	0	0	0
Y^{3}	0	R2	0	R6	0	0	0	0
Y^{4}	0	0	0	0	0	R2	0	R6
Y^{5}	0	R2	0	R6	0	0	0	0
Y^{6}	0	0	0	0	0	R2	0	R6
Y^{7}	0	R2	0	R6	0	R2	0	R6

TABLE 3.13
COMPARISON OF ADDERS OF DHT AND DCT IN [58]

scheme	Adder matrix	Adder bit-width
DCT	9 ALU +6	850
Proposed DHT	4 ALU +7	452

FPGA Implementation Results of 8-point 1-D DHT and comparisons

We have implemented the 8-points 1-D DHT in Xilinx FPGA using VHDL code and results are compared with ROM based DA of DHT implemented in Section 3.3. TABLE 3.14 shows the results and comparisons with the ROM based DA. ROM free DA implementation has less hardware requirement as compared to ROM based DA.

TABLE 3.14
HARDWARE UTILIZATION FOR PROPOSED DA FOR 1-D DHT

Logic Utilization	ROM based DA	ROM Free DA
# of Slices	561	309
# of 4 input LUTs	998	562
# Slice Flip Flops	341	0

2-D SDHT has been implemented using row-column decomposition technique and xc2vp30 device is programmed. Same sample 8x8 2-D data has been used here as before

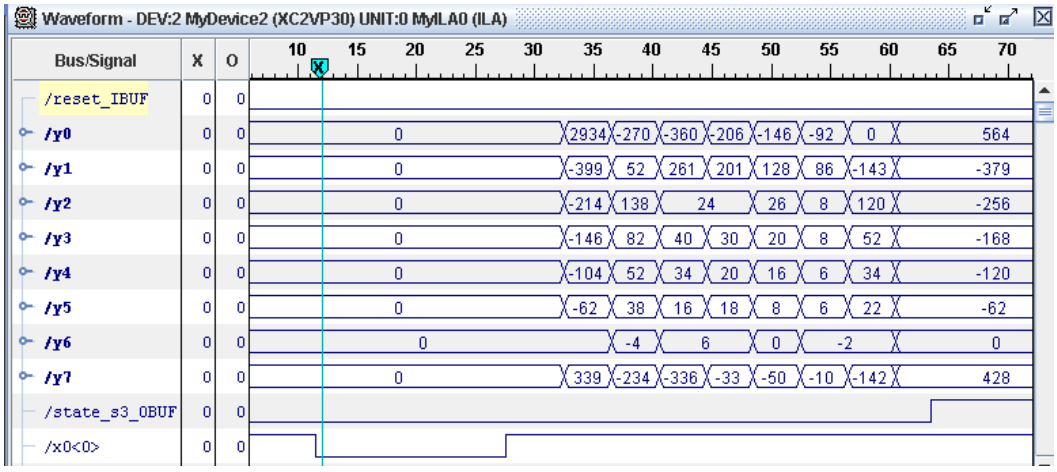


Fig. 3.16 Hardware implementation result of 8x8 image data matrix D_i by proposed DA for DHT method

in previous ROM based DA and Fig. 3.16 shows the 8x8 matrix output obtained using ChipScope pro device.

3.6 Conclusions

Different DSP based algorithm implementation approaches are described in this chapter. Two DA techniques, ROM based and ROM free, that implement multiply and accumulate operations in VLSI without using multiplier are illustrated. Using DA technique, 1-D DHT and 1-D DCT architectures are implemented and comparisons have been done. In both the cases (DCT and DHT), ROM based implementation has more area and power requirements along with slower speed. An area and power efficient DCT architecture is proposed in ROM free DA which reduces the number of computational modules in original ROM free DA. Implementation in standard cell based ASIC library of proposed architecture shows about 37% power savings and about 31% area savings in 8-point 1-D DCT architecture. FPGA implementation is also done and it shows the considerable FPGA resource reduction in proposed DCT architecture. Using row-column approach, 2-D DCT and 2-D SDHT are also implemented and hardware utilizations are summarized.

Chapter 4

Efficient JPEG Image Compression Architecture

4.1 Introduction

JPEG image compression is a standard image compression algorithm widely used for image compression in high end electronic circuits and systems as well as in battery powered devices such as digital camera. It uses steps in between 8x8 block wise DCT and binary stream storage in buffer memory such as quantization, zig-zag reordering and Huffman coding. To proceed from one step to another, memory is required to store block processed image data. Moreover, it uses Huffman code table for Huffman coding implementation where base code of DC and AC coefficients are stored as it makes hardware simple and high performing.

In this chapter, a simple hardware is presented for DCT to quantization to reduce the memory requirements in the intermediate stages by exploiting some of the flexibilities in JPEG implementation. Simulations have been performed to check the image quality by introducing the step to reduce the memory. Further, correctness of hardware implementation is demonstrated through comparisons of MATLAB and hardware outputs. The Huffman coding is implemented by employing the strategies to store the Huffman code table with reduced memory requirements.

4.2 Normalization matrix for hardware simplification in JPEG

JPEG coding procedure has been described in Section 2.6. 8x8 DCT, quantization and Huffman coding are three major steps followed in its implementation. Data from DCT output is quantized by a quantizer. Quantization is performed by dividing each DCT coefficient by a quantizer step size followed by rounding to nearest integer (Eq. 4.1).

$$F^{Q}(u,v) = Integer\ Round\left(\frac{F(u,v)}{Q(u,v)}\right) \tag{4.1}$$

where, $F(u,v)$ is the DCT coefficients and $Q(u,v)$ is the quantization matrix (also called normalization matrix). JPEG committee recommends to use a typical normalization matrix explained in Section 2.6, although users are free to use their own matrix. If that matrix is used for the quantization, then each coefficients must use a divider for the quantization along with 64 memory locations to store 64 quantization levels [19], [68–71]. A quantization matrix that can do the quantization without use of divider (or multiplier) and memory while maintaining the quality of image will be good choice for the hardware implementation. We have quantized the DCT coefficients in JPEG image compression by the following normalization matrix which is given as,

$$Q_n = \begin{bmatrix} 16 & 16 & 16 & 16 & 32 & 64 & 64 & 64 \\ 16 & 16 & 16 & 16 & 32 & 64 & 64 & 64 \\ 16 & 16 & 16 & 32 & 32 & 64 & 64 & 64 \\ 16 & 16 & 32 & 32 & 32 & 64 & 64 & 64 \\ 32 & 32 & 32 & 64 & 128 & 128 & 128 & 128 \\ 64 & 64 & 64 & 64 & 128 & 128 & 128 & 128 \\ 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \\ 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \end{bmatrix}$$

This matrix can further be scaled to achieve high compression, i.e.,

$$Q_n' = Q_n \times quality$$

where, quality is scaling parameter. Quantization using matrix Q_n can be performed by only shifting operations and this matrix is chosen by observing the fact that low spatial frequency contents, which are in top left region of DCT coefficients have the high visual information. Therefore, less quantization will preserve the image quality whereas bottom right regions of DCT coefficients (high frequencies) have very less visual information and can be discarded by high quantization. MATLAB simulations have been carried out for the performance comparisons of quantization matrix Q_n and the typical (normal)

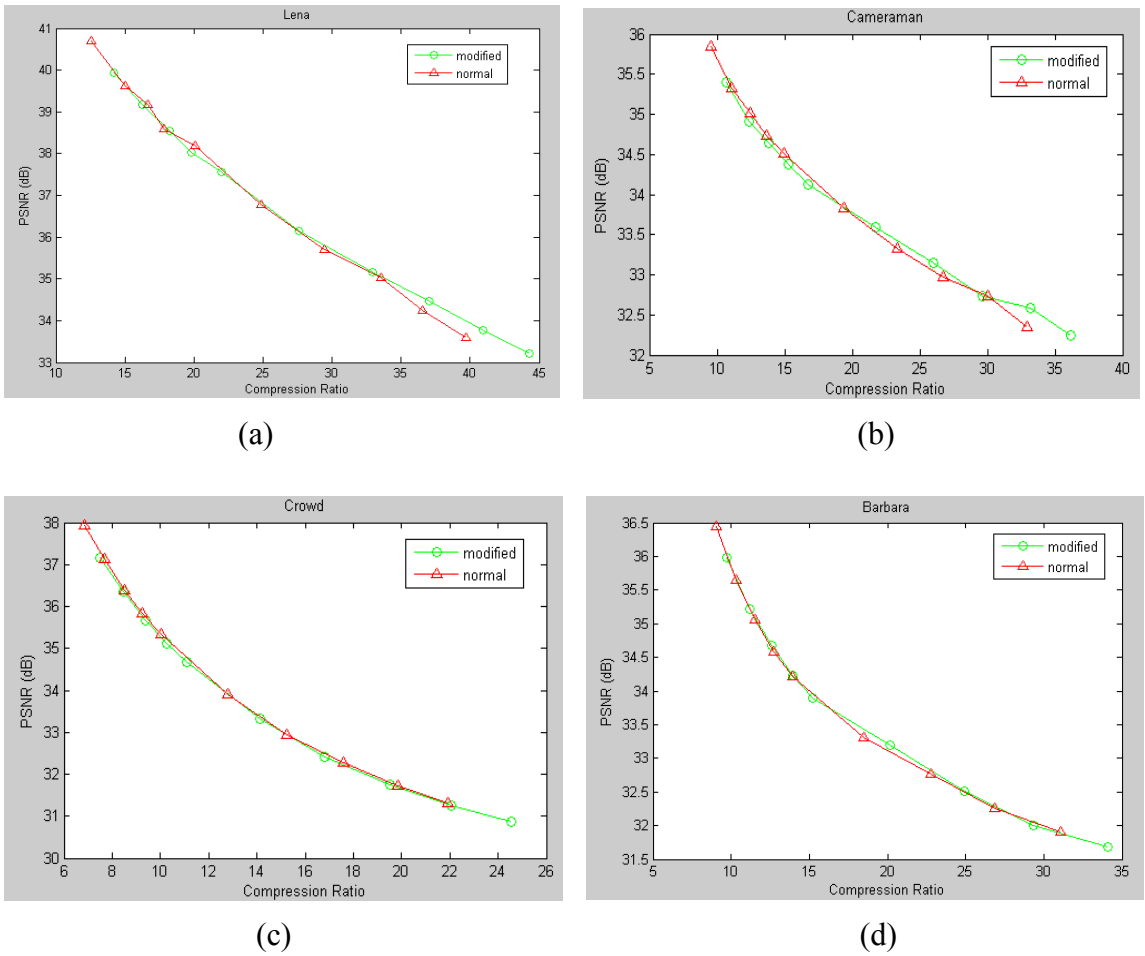
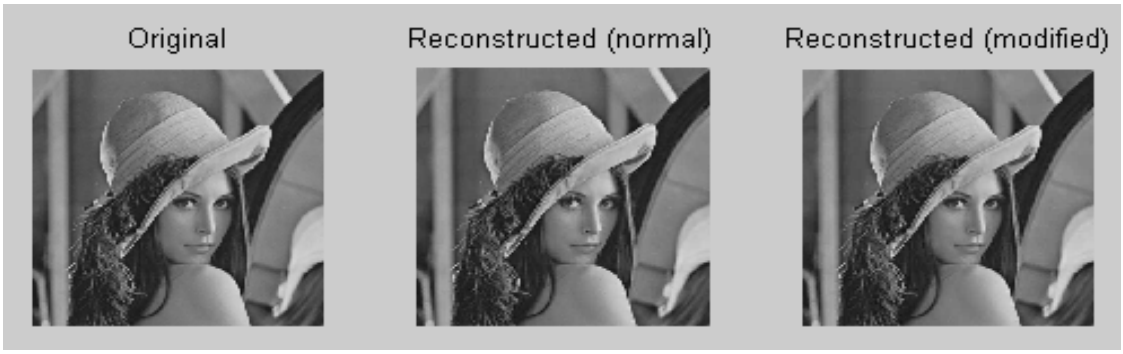


Fig. 4.1 PSNR against compression ratio for (a) 448x448 Lena, (b) 256x256 Cameraman, (c) 512x512 Crowd and (d) 512x512 Barbara Images

quantization matrix. PSNR against compression ratio for four standard images are plotted in Fig. 4.1. Fig. 4.2 shows the original and reconstructed images obtained by using both modified table and typical table (provided by JPEG) for normalization. Both the results (almost same PSNR and good visual qualities of images obtained) suggest that modified normalization matrix for the hardware simplification can be used in JPEG image compression. Moreover, in hardware implementation, there will be less round off error using modified table as it requires only bit shifting to perform quantization, whereas typical matrix will give more round off error because of fractional value obtained after division (fractional value representation has more accuracy at higher bit number of bits used for its representation).



(a)

(b)

(c)

(d)

Fig. 4.2 Original and reconstructed images using normal quantization matrix and modified matrix (a) 448x448 Lena, (b) 256x256 Cameraman, (c) 512x512 Crowd and (d) 512x512 Barbara

4.3 Efficient Architecture from DCT to Quantization and Re-ordering

An hardware efficient architecture for the computation of 2-D DCT, quantization and zig-zag ordering of the quantized coefficients is shown in Fig. 4.3. The 2-D DCT is computed by row-column decomposition method. When second 1-D DCT is being computed, timing and control will generate the eight addresses to store the eight transform coefficients each clock cycle in specified address (or register). For example, the first set of eight addresses generated will be 0, 2, 3, 9, 10, 20, 21, 35 (TABLE 4.1). Coefficients are stored in memory/registers after conditional shifting. First 8 outputs from the second 1-D DCT need to be divided by 16, 16, 16, 16, 32, 64, 128 and 128 for quantization which is done by shifting DCT coefficients right by 4, 4, 4, 4, 5, 6, 7, 8 and 8 bits. Similarly other coefficients from the second 1-D DCT are shifted according Q_n before storage in memory/registers. To synthesize the HDL code to have registers for storage, stored quantized coefficients, each at one clock cycle, are brought to output.

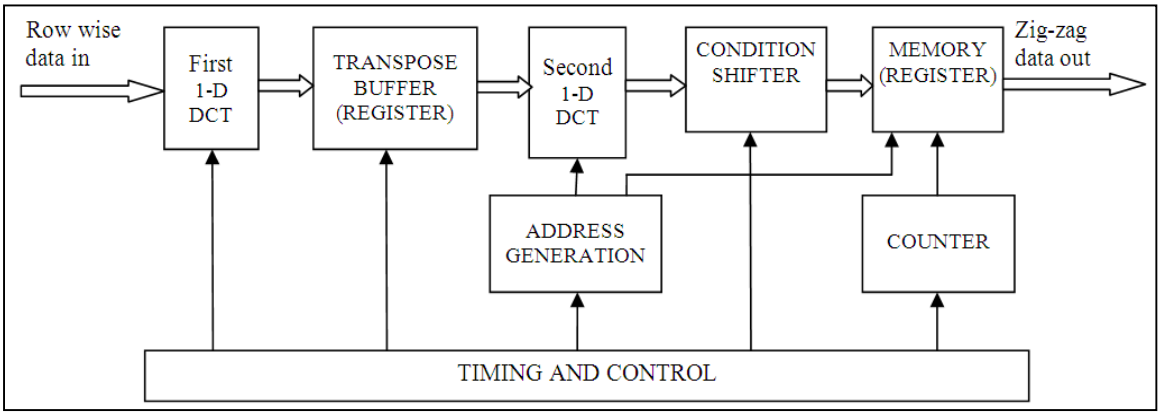


Fig. 4.3 DCT to Zig-zag re-ordering Architecture

TABLE 4.1
ZIG-ZAG ORDER SEQUENCE MATRIX

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

They are brought to the output in zig-zag order. Addresses for memory/registers are generated by 8-bits counter which counts from 0 to 63. This is because coefficients are stored in zig-zag order in increasing address of memory/registers from 0 to 63 which were generated for storage.

This architecture reduces a large number of memory/registers bits in different way. From 8-bits input data, 1-D DCT can produce 11-bits coefficients. These 1-D DCT coefficients when transformed to 2-D DCT by second 1-D DCT, can produce 14-bits output. If the quantization is performed after the 2-D DCT, then there is need of 64x14 bits of memory/registers to store 64 DCT coefficients. These bits are not required in the proposed architecture as outputs of second 1-D DCT are connected to the memory locations (or stored in registers) before shifting (quantization) as in Fig. 4.4. Only 64x10 bits of memory/registers are required to store quantized DCT coefficients completely, eliminating memory for the storage of 2-D DCT coefficients. Here, 10-bits of quantized coefficients are taken as minimum division is 16 (4-bits shift). Another memory/registers saving is achieved in the zig-zag ordering buffer. Instead of use of large memory for getting quantized coefficients in zig-zag re-order only 8 numbers of 8-bits registers are required to generate the 8 addresses in each clock cycle (TABLE 4.2). At each clock cycle, the contents of these 8 registers are changed by timing and control circuitry. The additional 8-bits counter is required for accessing quantized and stored coefficients in

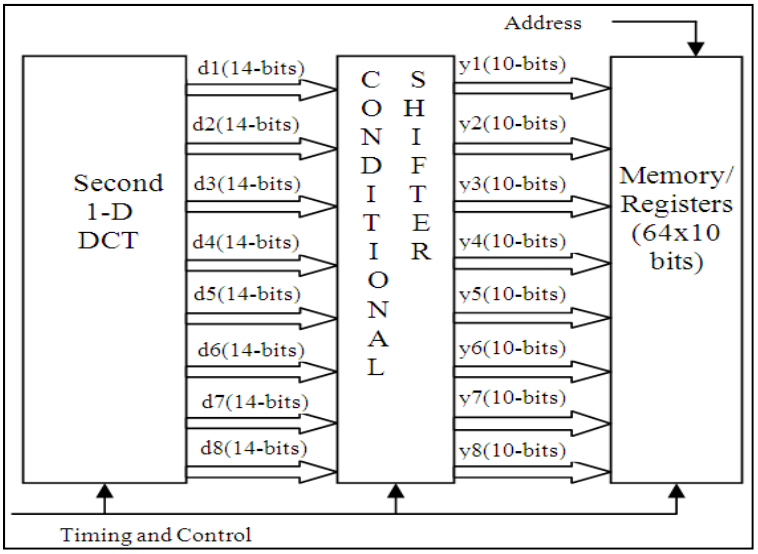


Fig. 4.4 2-D DCT Coefficients storage in 64x10 bits registers after shifting

TABLE 4.2
CLOCK CYCLE OPERATIONS FOR THE COMPUTATION OF 2-D DCT TO ZIG-ZAG ORDERING

Clock cycle	1	2	3	4	5	6	7	8	9	10	11		12		13		14		15		16		17		18	
											DCT Coeffs.	Address	DCT Coeffs.	Address	DCT Coeffs.	Address	DCT Coeffs.	Address	DCT Coeffs.	Address	DCT Coeffs.	Address	DCT Coeffs.	Address	DCT Coeffs.	Address
	BUFFER	1-D DCT of ROW 1	1-D DCT of ROW 2	1-D DCT of ROW 3	1-D DCT of ROW 4	1-D DCT of ROW 5	1-D DCT of ROW 6	1-D DCT of ROW 7	1-D DCT of ROW 8	BUFFER	y0	0	y0	1	y0	5	y0	6	y0	14	y0	15	y0	27	y0	28
											y1	2	y1	4	y1	7	y1	13	y1	16	y1	26	y1	29	y1	42
											y2	3	y2	8	y2	12	y2	17	y2	25	y2	30	y2	41	y2	43
											y3	9	y3	11	y3	18	y3	24	y3	31	y3	40	y3	44	y3	53
											y4	10	y4	19	y4	23	y4	32	y4	39	y4	45	y4	52	y4	54
											y5	20	y5	22	y5	33	y5	38	y5	46	y5	51	y5	55	y5	60
											y6	21	y6	34	y6	37	y6	47	y6	50	y6	56	y6	59	y6	61
											y7	35	y7	36	y7	48	y7	49	y7	57	y7	58	y7	62	y7	63

locations 0 to 63. TABLE 4.3 lists the total memory/registers bits and latencies required in literature by Agostini *et al.* [19] and proposed hardware architecture for the computation of quantization and zig-zag order only. A total of 1336 bits have been saved in quantization and zig-zag ordering along with 896 bits in 2-D DCT coefficients storage (TABLE 4.4). Because both implementation platforms are different, logic cells cannot be compared. But from the simplicity of the proposed architecture, hardware savings can be predicted which is further shown in implementation details in next Section where FPGA implementation details and hardware utilization is shown. For the scaling to get more compression, quantization step-size can be changed by adding one multiplier at the output of zig-zag buffer without changing any internal hardware.

TABLE 4.3
MEMORY BITS AND LATENCY COMPARISONS FOR THE QUANTIZATION AND ZIG-ZAG BUFFER IN PROPOSED HARDWARE SCHEME WITH EXISTING

	Memory bits	Latency
Agostini et al. [19]	2048 (768+1280)	70
Proposed Hardware	712	72

TABLE 4.4
MEMORY/REGISTERS SAVINGS ACHIEVED IN PROPOSED DCT TO ZIG-ZAG ARCHITECTURE

	In Quantization and Zig-zag buffer	In storing 2-D DCT Coeffs.
No. of Bits saved	1336	896

FPGA/ASIC implementation results and discussions

The 1-D DCT implementation is performed for the compressed DA based algorithm proposed by Chungan *et al.* [58] (explained in section 3.4.1), but shifting is performed by division operator (/) in VHDL code to reduce the error due to sign extension. Quantization is performed by both wiring (each LSB removal is equal to division by 2) and division operator in two different implementations. 8-bits input, 14-bits internal word representation and 12-bits DA precision have been used for the implementation. Implementation is carried out in Xilinx XC2VP30 FPGA device in Virtex-II Pro board as well as Synopsys DC using TSMC CLN65GPLUS 65 nm technology library. FPGA implementation results are tabulated in TABLE 4.5. Total of 3070 slices are used in FPGA implementation of DCT to zig-zag order when division is performed by using operator (/) whereas wiring to shift for quantization uses 2856 slices. From the results of 2-D DCT implementations, it can be seen that quantization and zig-zag ordering are realized by additional 635 slices and 633 registers which are very less as compared to previous literature mentioned in previous section. Register savings lead to low power consumption as well as area savings. The total cell area in the synthesis of VHDL code in Synopsys DC is 30527.64 μm^2 in only 2-D DCT and 39015.36 μm^2 in DCT to zig-zag. TABLE 4.6 shows the Synopsys DC implementation results for total cell area and power consumption in different implementations.

TABLE 4.5
HARDWARE UTILIZATION POWER DISSIPATION FOR DCT TO ZIG-ZAG ORDERING ARCHITECTURE IMPLEMENTED IN FPGA

FPGA Chip: Xilinx XC2VP30					
	# of 4 input LUTs	# of slices	# of slice Flip Flops	Clock Freq. (MHz)	Power (W)
Only 2-D DCT	4502	2435	868	48.4	14.97
Zig-zag ordered using division operator for quantization	5276	3070	1501	31.1	16.58
Zig-zag ordered using wire shifting for quantization	4986	2856	1409	40	16.53

TABLE 4.6
HARDWARE UTILIZATION POWER DISSIPATION FOR DCT TO ZIG-ZAG ORDERING ARCHITECTURE IMPLEMENTED IN ASIC LIBRARY

TSMC CLN65GPLUS 65 nm technology (clock frequency = 500 MHz)			
	Total cell area	Total Dynamic Power (global operating voltage 1.1v)	Min. Slack at 500MHz
Only 2-D DCT	30527.64 μm^2	6.78 mW	0.477 ns
Zig-zag ordered using division operator for quantization	39015.36 μm^2	10.05 mW	0.01 ns
Zig-zag ordered using wire shifting for quantization	38133.30 μm^2	10.07 mW	0.059 ns

Functional Verification through simulation and Hardware results

For the functional verification, a sample 8x8 image data in the range -128 to 127 has been taken which is given by,

$$D_s = \begin{bmatrix} -95 & -96 & 99 & 98 & 94 & 100 & 57 & 54 \\ 108 & 103 & 99 & 98 & 94 & 80 & 57 & 39 \\ 107 & 102 & -2 & -3 & 94 & 80 & 58 & -60 \\ 104 & 99 & 96 & 96 & -7 & -20 & 58 & 41 \\ 2 & -3 & -6 & -5 & -7 & -19 & -41 & -58 \\ -1 & -5 & -8 & -6 & -8 & -19 & 60 & 42 \\ -3 & -7 & -9 & -7 & -8 & -19 & -40 & -57 \\ -4 & -7 & -10 & -8 & -8 & -19 & -40 & -57 \end{bmatrix}$$

and its 2-D DCT MATLAB output is shown in Fig. 4.5. After quantizing the DCT coefficients by the modified quantization table followed by Zig-zag ordering, the coefficients are shown in Fig. 4.6. Prototyping on Xilinx Virtex-II FPGA board for the implementation of shifting done by division operator architecture, 2-D DCT outputs obtained through ChipScope pro is shown in Fig. 4.7. Fig. 4.8 shows the quantized and zig-zag ordered coefficients. 25 MHz clock frequency (obtained through DCM) is used as synthesized deign shows maximum frequency of 31.1 MHz. Small error is observed in hardware implementation is because of truncation in the number of bits representation.

180.8750	59.9902	-60.9457	21.3503	-13.6250	30.0316	-12.2333	18.4493
254.5078	-45.8502	-69.5624	3.4947	-11.1100	9.9324	24.2949	12.0433
-21.5291	-92.8626	-106.3129	-9.3646	4.8728	-7.4862	56.3815	8.5475
-75.3297	-142.5989	-49.3253	-77.6570	15.9887	22.8467	44.1011	12.7017
-58.8750	-28.2521	-52.1888	-79.5015	11.6250	30.9812	39.6121	-6.0567
1.8608	-29.1632	-13.1193	-34.7535	-4.1999	26.9269	5.1404	4.1143
-27.4778	-81.0117	2.1315	43.1987	-17.3072	-4.1892	-16.6871	22.6674
-122.8568	4.9432	-63.6263	94.3544	-14.8274	-40.5439	11.7990	6.0803

Fig. 4.5 MATLAB Simulation results for 2-D DCT of sample data D_s

11	4	16	-1	-3	-4	1	-4
-6	-5	-2	-9	-7	0	0	0
0	0	-2	-1	0	0	0	-2
-2	0	0	0	0	0	0	0
-1	0	-1	-1	0	0	-1	0
0	1	0	0	1	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Fig. 4.6 Quantized and zig-zag ordered coefficients of D_s (arranged in left to right and top to bottom order)

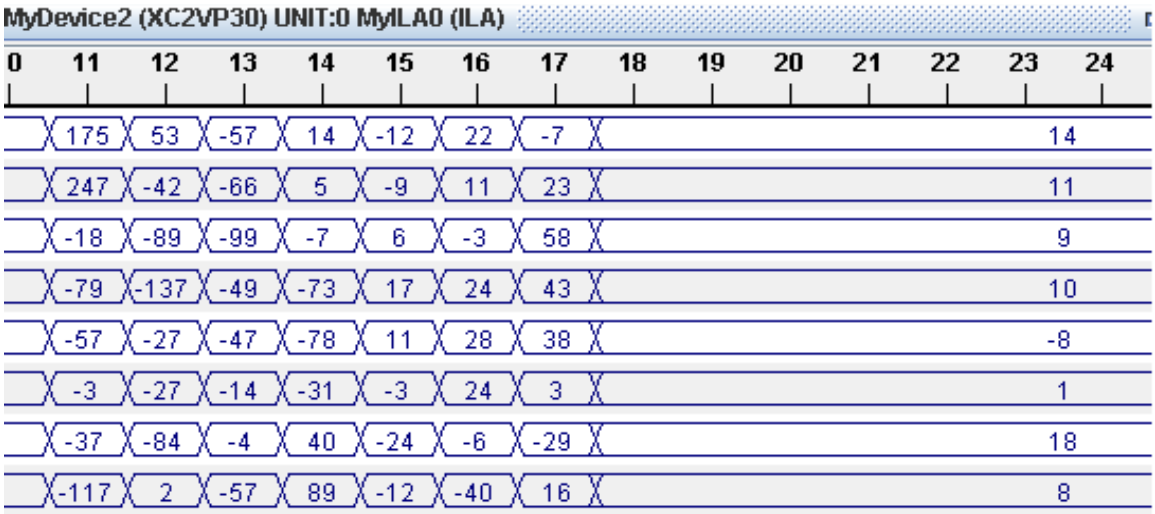


Fig. 4.7 2-D DCT coefficients of D_s obtained through Xilinx ChipScope Logic Analyzer

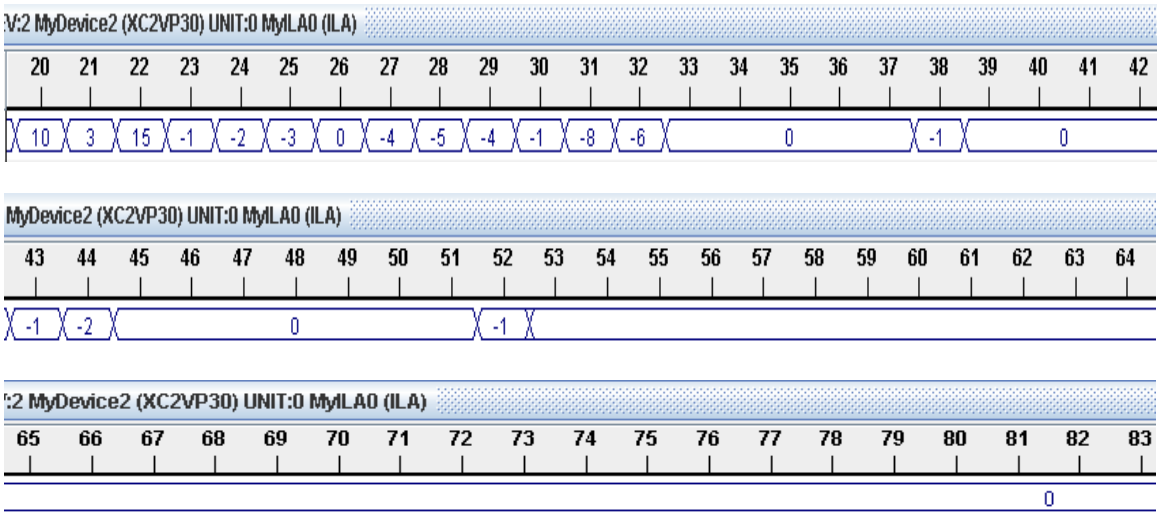


Fig. 4.8 Quantized and Zig-zag ordered 2-D DCT coefficients of D_s obtained through Xilinx ChipScope Logic Analyzer

4.4 Huffman Coding Architecture Implementation in FPGA for JPEG

Huffman coding is a variable length code used in image compression for the removal of data redundancy. It is done by encoding more frequent occurring symbols (data) with less number of bits and less frequent symbols with more number of bits. Optimized code for the data is obtained. For the hardware implementation, use of Huffman code table makes the hardware simple and high performing [19]. JPEG uses a code table (called Huffman code table) to do the Huffman coding [1, 15, 69–71]. The architecture of Huffman coding has been explained in Section 2.6 (JPEG baseline image coding). Since FPGA has more dedicated memory (RAMs/ROMs), it is a good choice for Huffman coding implementation. The following steps are to be carried out for the implementation after quantization of DCT coefficients.

1) Storing Huffman code tables for DC and AC coefficients separately in memory
2) Category selection
3) Bringing the DC coefficient difference base code from the DC base code table
4) Extending the DC base code with binary value of DC difference coefficient
5) Bringing the AC coefficient base code from the AC base code table
6) Extending the AC base code with binary value of AC coefficient
7) Repetition from step 5 until all AC coefficients are encoded

The Huffman code tables for DC and AC coefficients are given in Appendix A and Appendix B respectively. DC code table is small and coding is done simply by addressing the proper location in DC base code memory whose address is given by category of the coefficient. TABLE 4.7 shows the category for different DCT coefficients range. The AC coefficient base code table is addressed by an additional variable namely run. It is the number of zeros preceding the particular AC coefficient. If there are 16 zeros preceding a coefficient, it is encoded by a special code whose value is "111111110111". The end of block is encoded using "1010". Fig. 4.9 shows the sequence of code formed by the encoding of DCT coefficients according to JPEG Huffman code table. The code lengths of AC base code are variable and therefore efficient storage of code tables in memory can save the memory space. Sun *et al.*[71] have constructed the efficient way of memory for AC coefficients storage. They have used 8-bits for each (run, cat.) pair storage. In our design, only 7-bits are used for each (run, cat.) savings 160 memory bits. Fig 4.10 shows RTL schematic of Huffman coding algorithm implemented in Xilinx FPGA. It consists

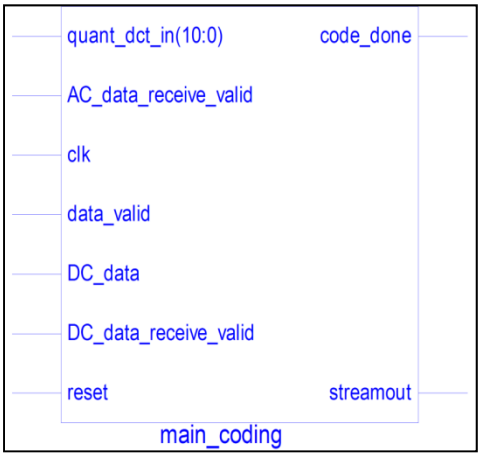
TABLE 4.7
CATEGORY OF DCT COEFFICIENTS

Category	Quantized DCT Coefficients Range
0	-
1	-1, 1
2	-3, -2, 2,3
3	-7,...-4, 4,...7
4	-15, ...-8, 8, ...15
5	-31, ...-16, 16,...31
6	-63, ...-32, 32, ...63
7	-127, ... -64, 64, ... 127
8	-255, ...-128, 128, ...255
9	-511,... -256, 256, ...511
10	-1023, ...-512, 512, ...1023
11	-2047,... -1024, 1024,... 2047

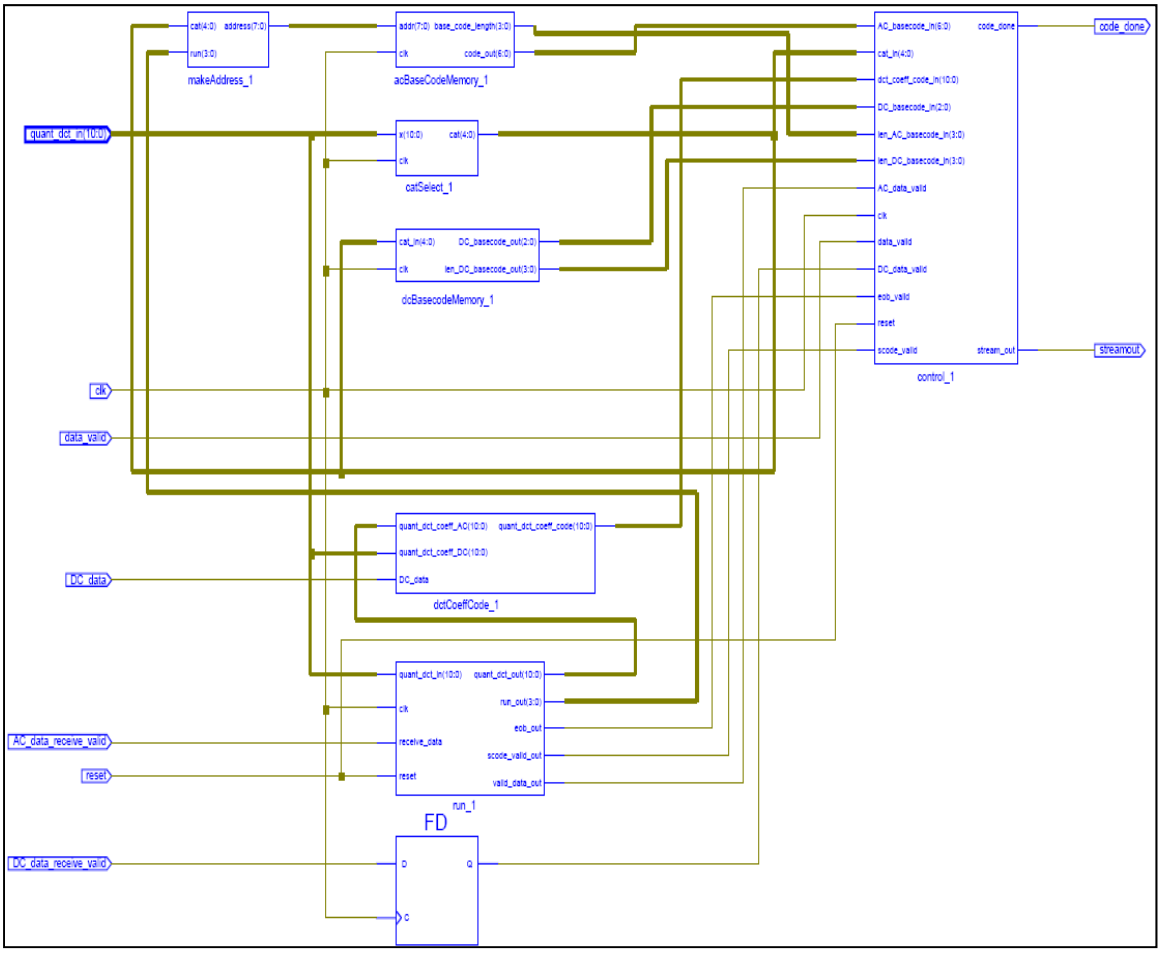
DC base code	Binary value of DC Coeff.	AC base code/Special Code ("11111110111")	Binary value of AC Coeff.	• • •	1010

Fig. 4.9 Coding Sequence of DCT coefficients in JPEG

RTL Schematic of Huffman Code table in Xilinx FPGA



(a)



(b)

Fig. 4.10 RTL Schematic of Huffman Coding implemented in Xilinx FPGA (a) Top module and (b) Detail schematic

of the following 7 individual modules.

1) Category selection module (catSelect_1)
2) Memory module for storing DC base code (dcBasecodeMemory_1)
3) DCT coefficient code module (dctCoeffCode_1)
4) Run module (run_1)
5) Address formation module (makeAddress_1)
6) Memory module for storing AC base code table (acBaseCodeMemory_1) and
7) Control module (control_1)

1) Category Selection Module

The category selection module takes input as 11-bits quantized DCT (represented by x) coefficients and gives the category of the coefficient as output in the rising clock edge. It is implemented by a simple logic which detects position of last '1' from LSB occurring in the input data. The position number is the category. The negative data input is converted into the equivalent positive number before finding the category. Fig. 4.11 shows the tip level interface of the circuit and Fig. 4.12 shows the simulation output result in Xilinx ISE 10.1.

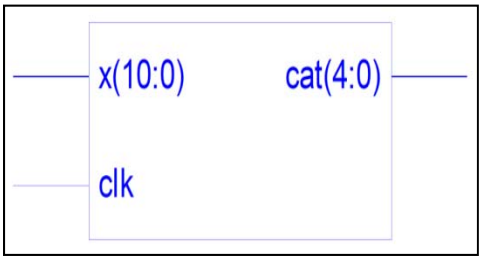


Fig. 4.11 Top Level Interface of Category selection module

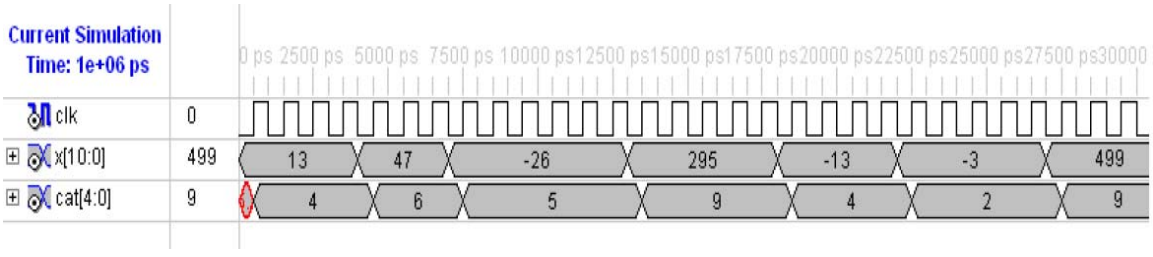
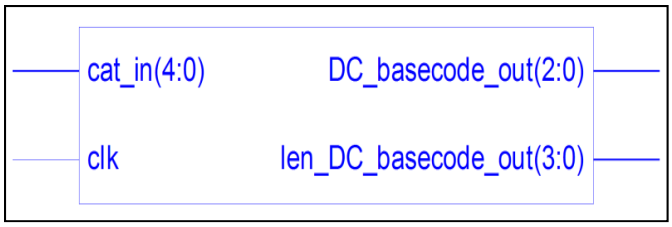


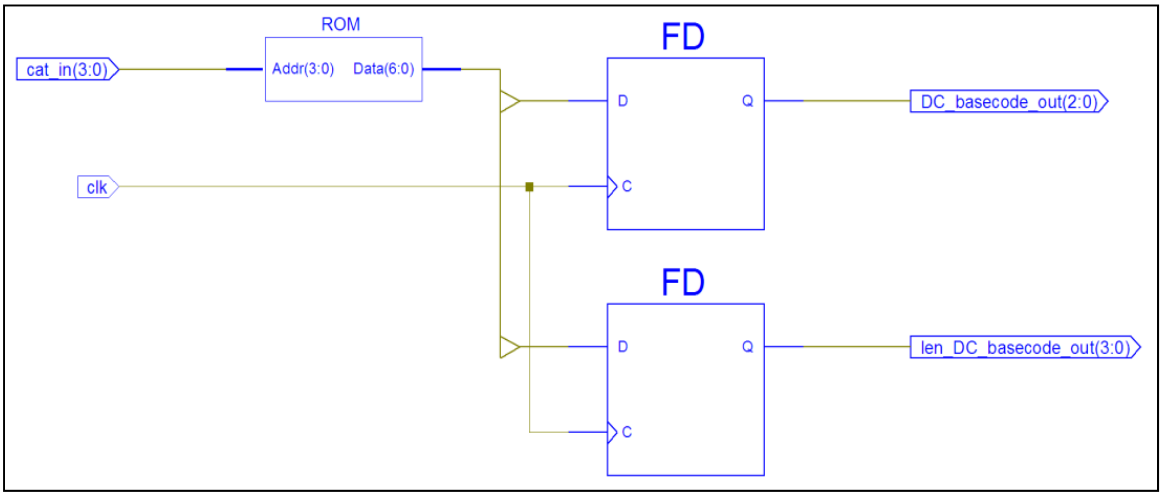
Fig. 4.12 Simulation output of Category selection module

2)Memory Module for storing DC base code

It consists of two different ROMs where DC base code and its length are stored. Size of base code ROM is12x3-bits and size of the its length is 12x4-bits. It takes the category of the DC coefficients and gives the DC base code and its length simultaneously in the rising clock edge. The length is required when forming the final code. Fig. 4.13 shows the RTL schematic and Fig. 4.14 shows the simulation output.



(a)



(b)

Fig. 4.13 RTL Schematic of DC base code module (a) Top interface and (b) Details

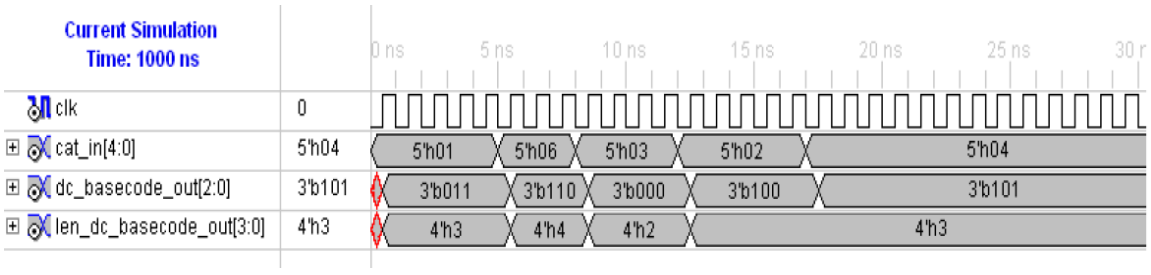
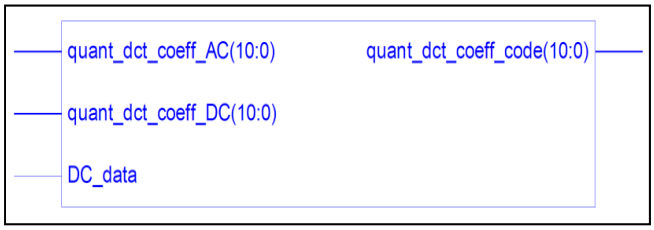


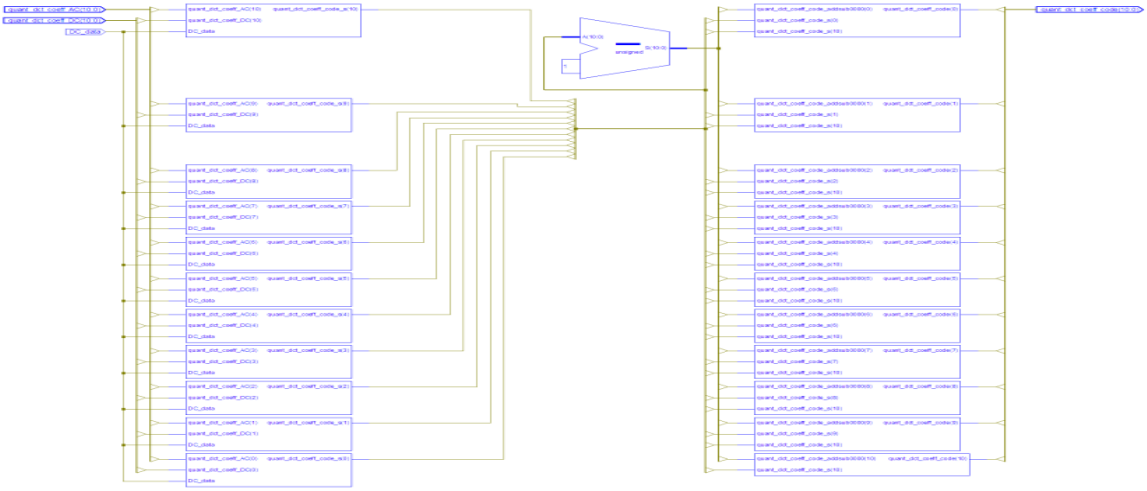
Fig. 4.14 Simulation result of DC base code module

3)DCT coefficient Code Module

DCT coefficient code module makes the proper code of the data obtained after quantization. Fig. 4.15 shows the RTL schematic. It is performed by subtracting '1' from the input data if MSB is '1' otherwise same (input) data is the output. One subtractor is used here. DC_data is a controlling signal input which selects quantized DC coefficient as input if its value is '1' else it selects AC coefficients. Fig. 4.16 shows the simulation result.



(a)



(b)

Fig. 4.15 RTL Schematic of DCT Coefficient code module (a) Top interface and (b) Details

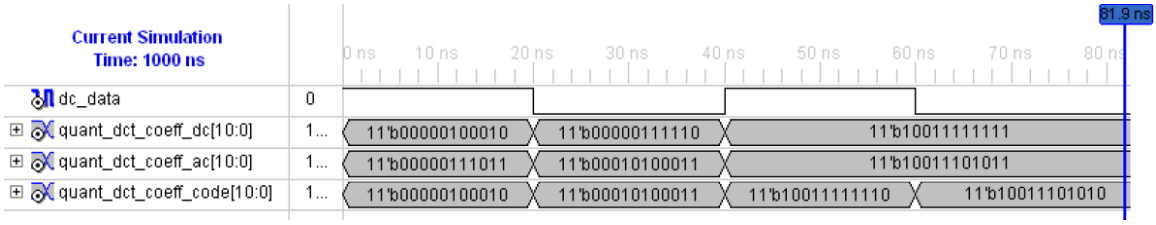


Fig. 4.16 Simulation result of DCT coefficient code module

4)Run Module

It takes the quantized DCT coefficients, counts the number of zero coefficients received and gives the same output if non-zero coefficient is received with the number of zeros (‘run_out’) preceding this coefficient and the a valid data signal (‘valid_data’) indicating that the current coefficient is a non-zero. Input data to this module is recognized by ‘receive_data’ signal. Special code valid signal ‘scode_valid_out’ is inserted ‘1’ when non-zero coefficients received in a row exceeds 16. A counter counts the total coefficients received. If count reaches 63 (all current block AC coefficients have been received), an end of block signal ‘eob_out’ is inserted high. Fig. 4.17 shows the top level view of RTL schematic and Fig. 4.18 shows the simulation result.

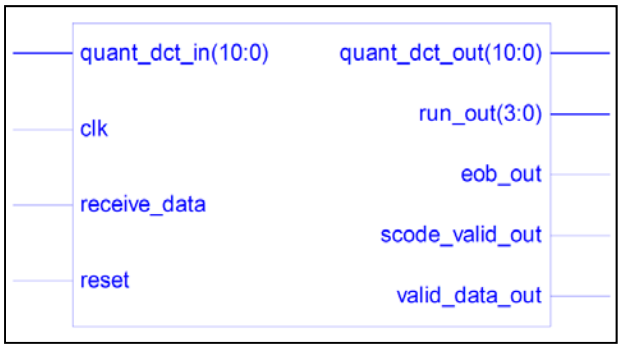
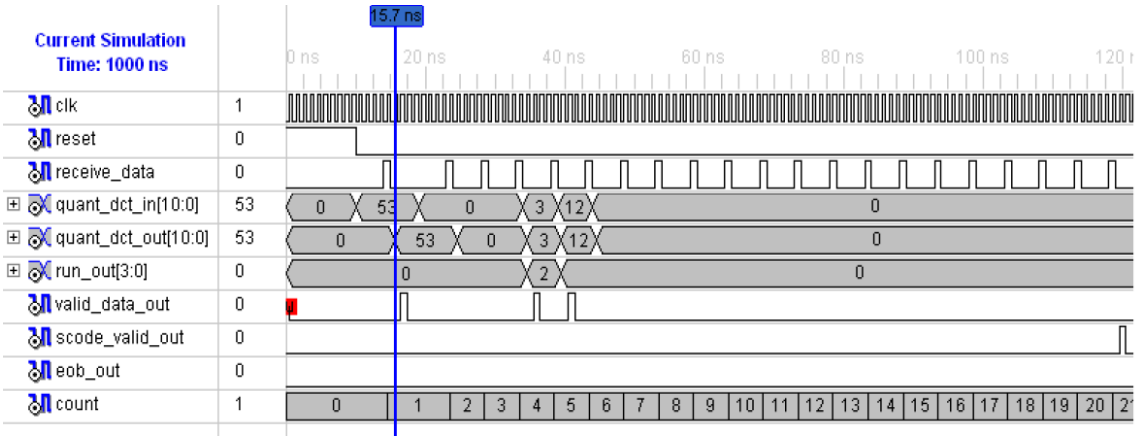
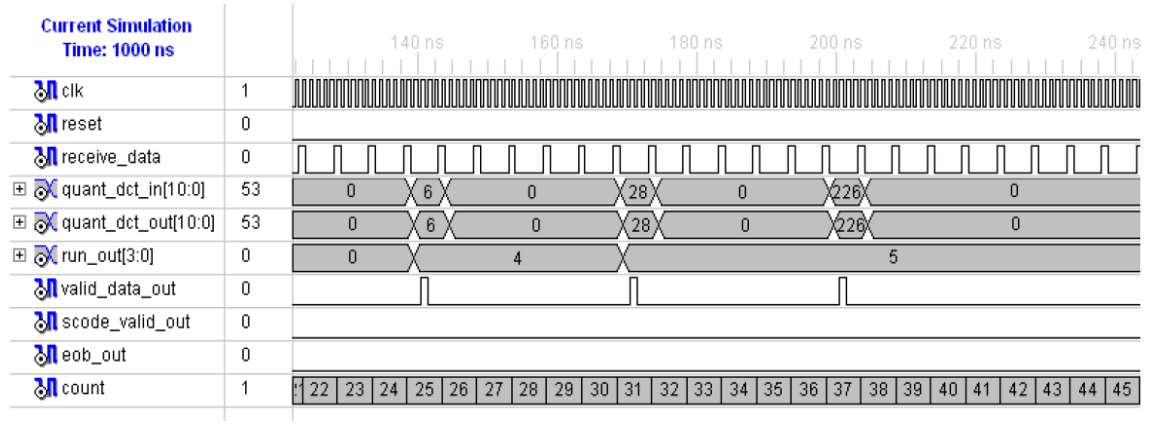


Fig. 4.17 RTL Schematic of Run Module (top view)

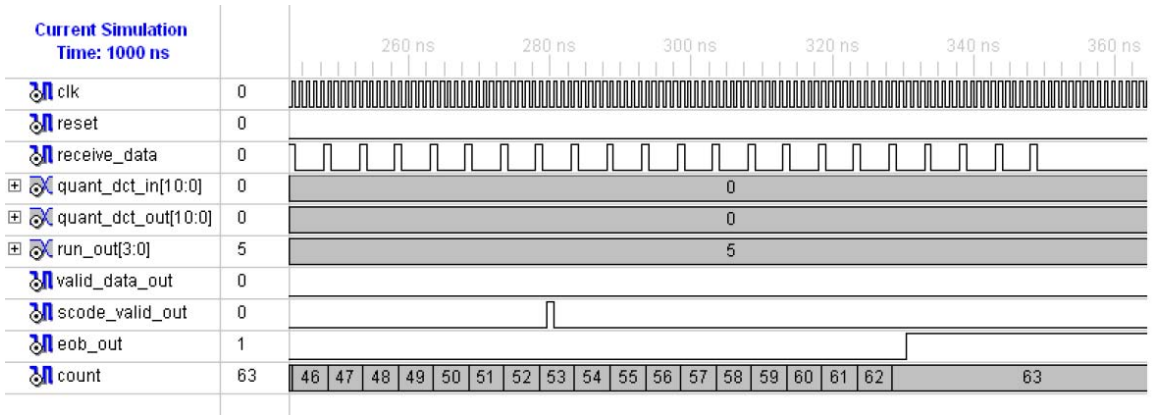


(a)

Fig. 4.18 Simulation result of run module for received AC coefficients (a) 1 to 21



(b)



(c)

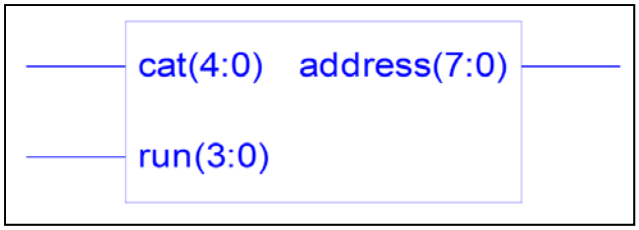
Fig. 4.18 (continued) (b) 22 to 45 and (c) 46 to 63

5)Address formation Module

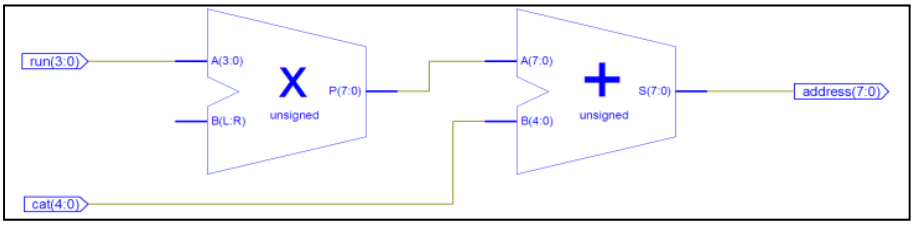
The AC base code table is stored in continuous memory locations from 0 to 161. Address formation module takes 'run' (from run module output) and 'cat' (from category selection module) of the AC coefficients and gives the 8-bits 'address' (as total of 161 locations in AC base code table have to be addressed) as output for the AC base code table. It is an asynchronous circuit as the address output should available as soon as 'cat' input changes. Each additional 'run' increments the 'address' by 10 times and each additional 'cat' increments it by one location, i.e.,

$$address=(run \times 10) + cat$$

Fig. 4.19 shows the RTL schematic and Fig. 4.20 shows the simulation results.



(a)



(b)

Fig. 4.19 RTL Schematic of Address formation module (a) top view (b) detail view

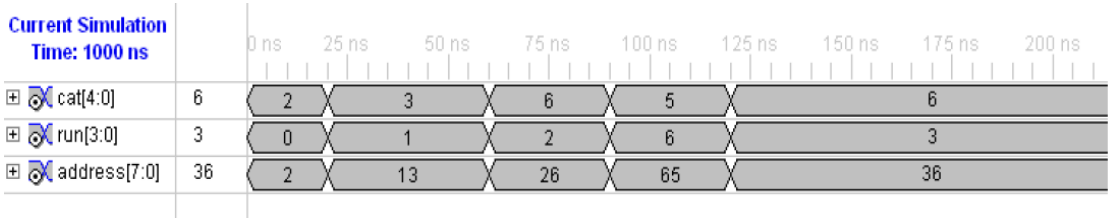


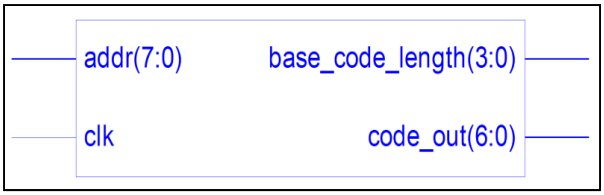
Fig. 4.20 Simulation result from address formation module

6)Memory Module for Storing AC base Code Table

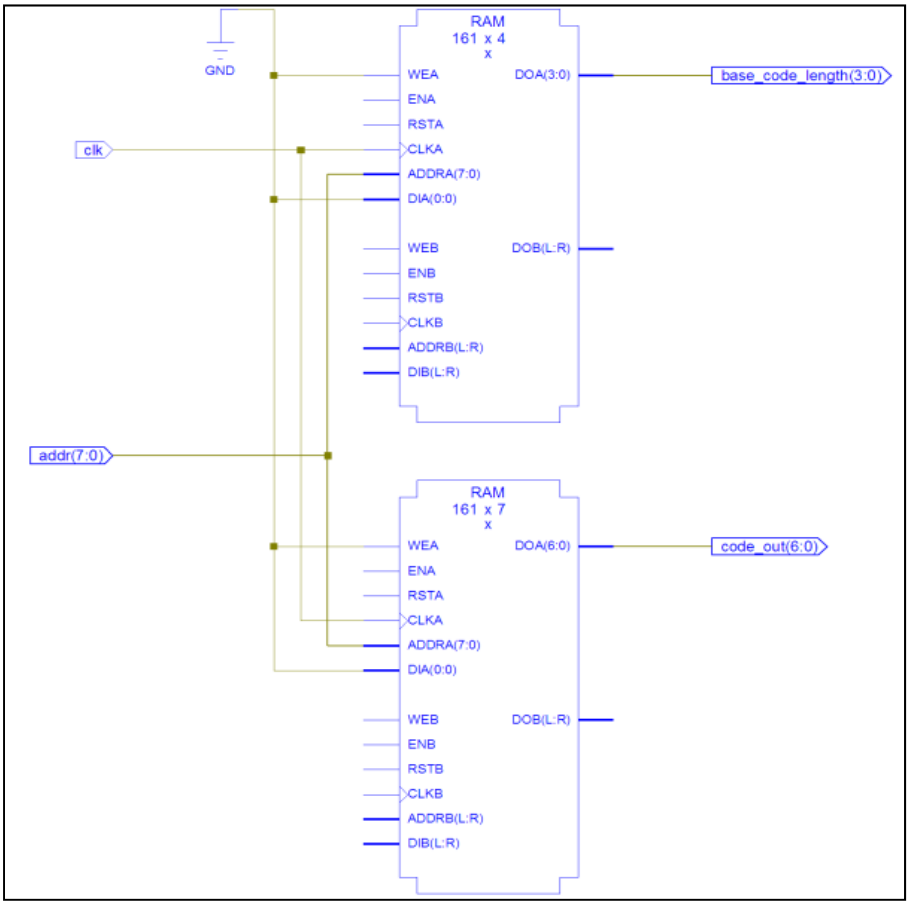
There are two ROMs used for the implementation (shown in Fig. 4.21). One is having size 161x7 and it stores the AC base code. Here, 7-bits storage have been used as a maximum of 7-bits (right most) out of 16-bits length are different. Left most 9-bits are either not required or are all '1's (see Appendix B). The more than 7-bits base code required is extended by adding extra '1's to it. For example, if base code is "1111110011" (10-bits) then "1110011" is stored in memory, i.e.,

Complete AC base code= "111" (extension bits) & "1110011" (from ROM)

The second ROM (size 161x4) stores the code length of the corresponding AC base code. The module takes the input address from the address formation module and gives base code as output and its length at rising clock edge (shown in Fig. 4.22).



(a)



(b)

Fig. 4.21 RTL Schematic of Memory module for AC base code storage (a) top view and (b) detail view

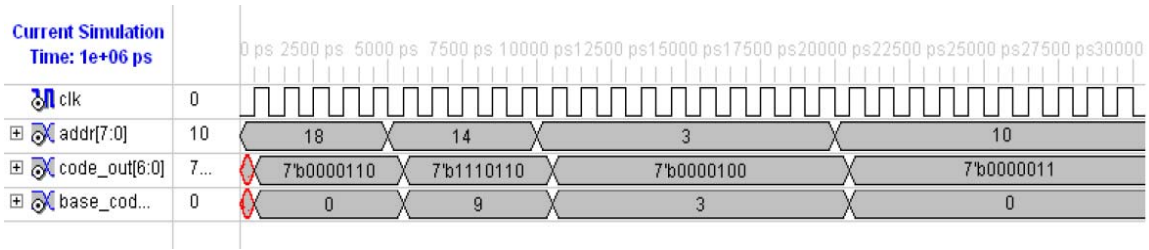


Fig. 4.22 Simulation result from AC base code memory module

7)Control Module

The control module controls all the coding activity. It is implemented using FSM. The top interface is shown in Fig. 4.23. The state of the FSM changes according to the input control data received. There is a buffer register named 'buff_reg' of bit-widths 201 where all the codes received are buffered. The initial state of the controller is 'init' at reset and on receiving 'data_valid' signal, it starts buffering of data in 'buff_reg' register. If 'DC_data_valid' control signal is received, it goes to 'dcbase' state. In this state, it takes the extended DC base code data (in 'dcbasecode' register) at each clock cycle as shown in Fig 4.24(a). The extended 9-bits DC base code is formed by taking 3-bits DC base code from DC base code memory through 'DC_basecode_in' interface and its length through 'len_DC_basecode_in' interface. Soon after completion of base code, the FSM goes to 'dccoeff' state where it stores the binary value of DC coefficient having length equal to its category. The total number of data stored in the buffer register is shown through 'code_len_count' register status. At the end of 'dccoeff' state, the value of 'code_len_count' is 9 as it has 5-bits as DC base code and 4-bits as DC coefficient code value.

On receiving 'AC_data_valid' signal high, the controller goes to AC base code state 'acbase' followed by AC coefficient code state 'accoeff' (shown in Fig. 4.24(b)). The same thing is done here as described for DC base code and DC coefficient code. The length of AC base code received from memory (through 'AC_basecode_in' bus) is 7-bits and it is extended by adding '1's to form the total length 16 in 'ext_AC_basecode_in' register. However, only required length obtained through 'len_AC_basecode_in' is taken into consideration for storing bits in buffer register.

When 'scode_valid' signal is inserted high, it goes to 'acspl' state and stores 12-bits special code ("11111110111") in buffer register in 12 clock cycles (Fig. 4.24(c)). When all AC coefficients are encoded, which is indicated by inserting 'eob_valid' signal high, the controller goes to state 'eob' (Fig. 4.24(d)) and it stores "1010" bit streams. At the end of this state, 'code_done' signal is inserted high. This state is followed by data output state 'dataout' where buffered data are brought to the output. Here, one bit data per clock cycle is considered for the implementation, although desired number of bits per clock cycle can be taken to output depending on throughput requirements.

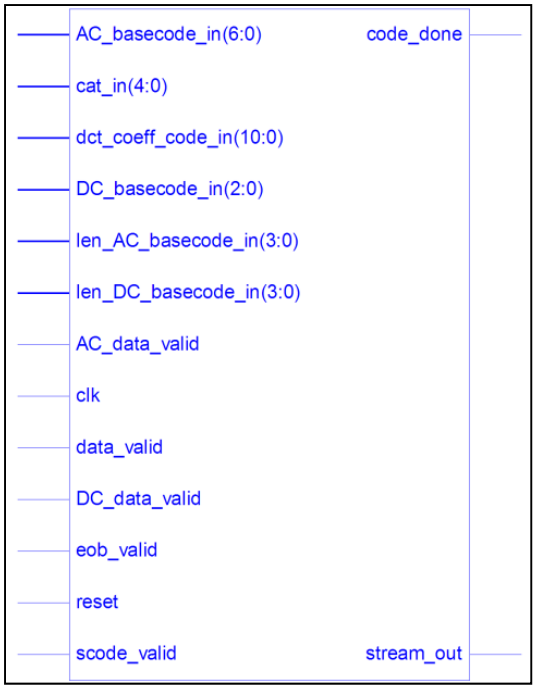
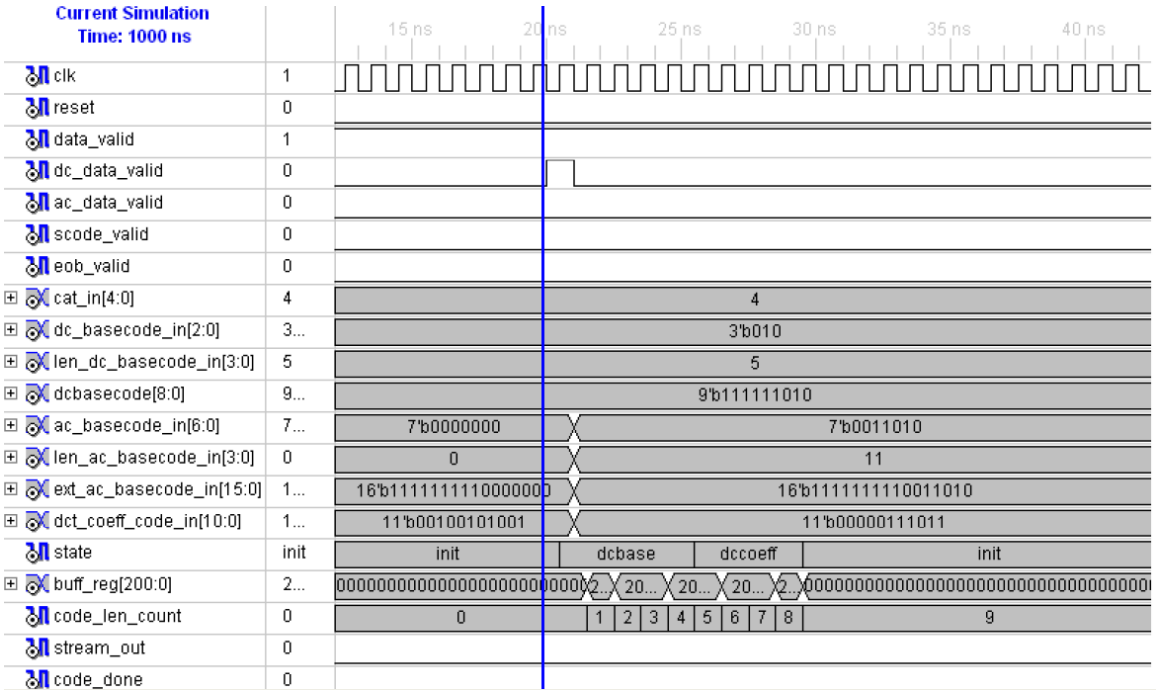
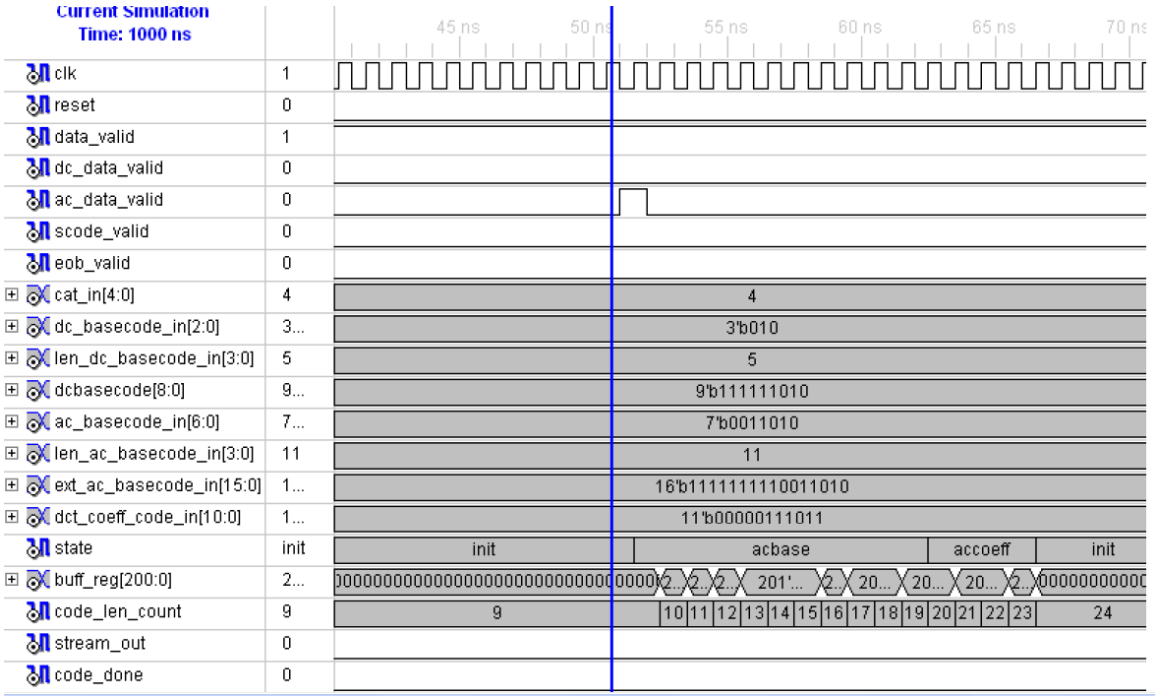


Fig. 4.23 RTL Schematic of control module (top view)

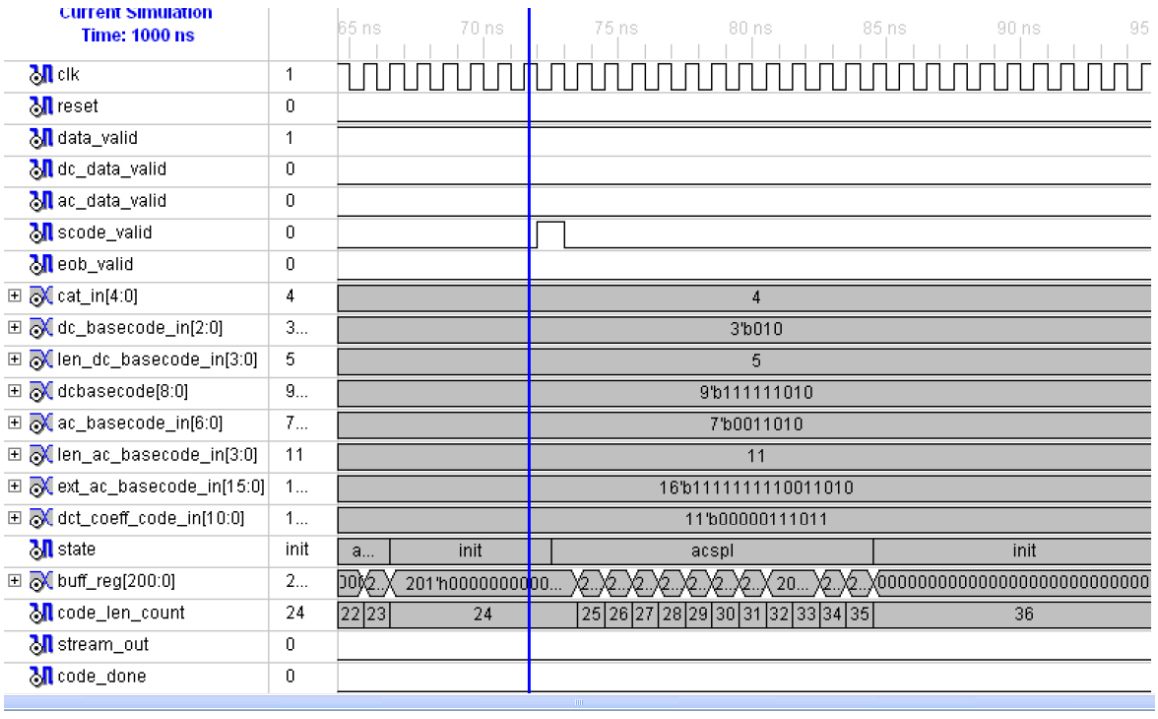


(a)

Fig. 4.24 Simulation results of control module (a) DC coefficient coding

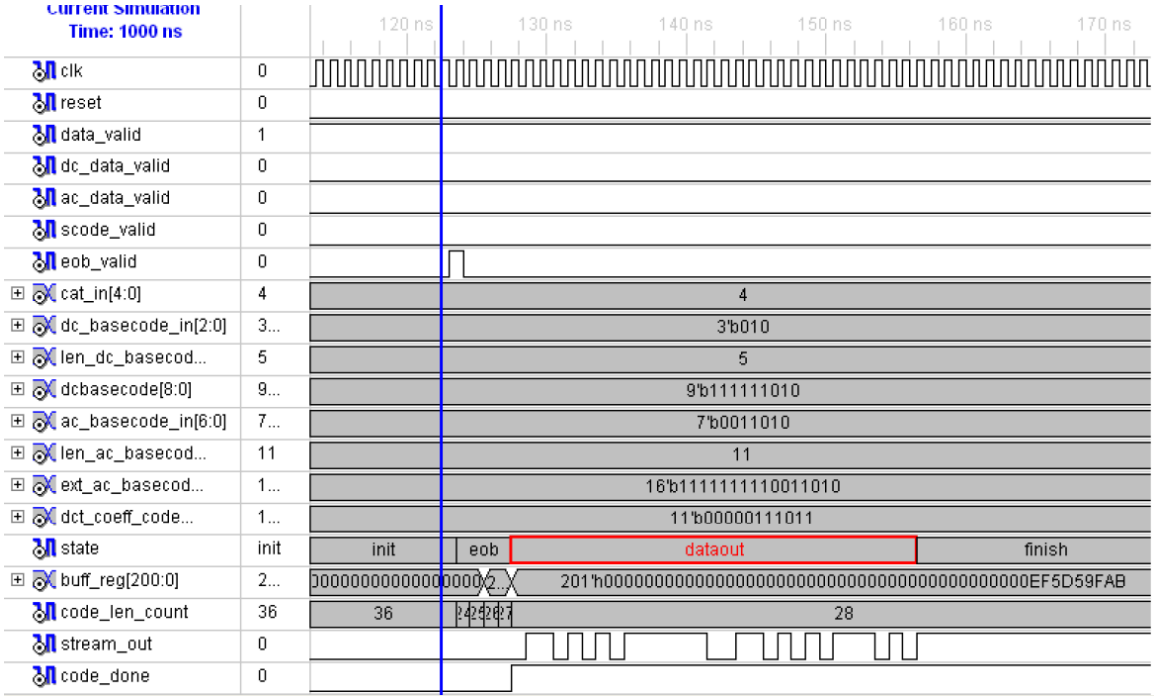


(b)



(c)

Fig. 4.24 (continued), (b) AC coefficient coding, (c)Special code



(d)

Fig. 4.24 (continued), (d) Buffered Output as bit stream

Design Summary of Huffman Coding Implementation

Fig. 4.25 shows the advanced HDL synthesis report of the complete Huffman coding

```
Advanced HDL Synthesis Report

Macro Statistics
# RAMs                                          : 2
 161x4-bit single-port block RAM                : 1
 161x7-bit single-port block RAM                : 1
# ROMs                                          : 3
 12x1-bit ROM                                   : 1
 12x7-bit ROM                                   : 1
 4x1-bit ROM                                    : 1
# Multipliers                                   : 1
 4x4-bit multiplier                             : 1
# Adders/Subtractors                            : 7
 10-bit adder                                   : 1
 11-bit subtractor                              : 1
 5-bit adder                                    : 1
 5-bit subtractor                               : 1
 8-bit adder                                    : 2
 8-bit addsub                                   : 1
# Counters                                      : 1
 6-bit up counter                               : 1
# Registers                                     : 267
 Flip-Flops                                     : 267
# Comparators                                   : 1
 8-bit comparator equal                         : 1
# Multiplexers                                  : 4
 1-bit 11-to-1 multiplexer                      : 1
 1-bit 16-to-1 multiplexer                      : 1
 1-bit 201-to-1 multiplexer                     : 1
 1-bit 9-to-1 multiplexer                       : 1
```

Fig. 4.25 Macro Statistics of Advance HDL Synthesis of Complete Huffman Coding

TABLE 4.8
DESIGN SUMMARY OF HUFFMAN CODING IMPLEMENTED IN FPGA

FPGA Chip: Xilinx XC2VP30						
Module	# of Slices	# of Slice Flip Flops	# of 4 i/p LUTs	Max. Freq.(MHz)	$Dynamic Power (mW)	BRAMs
Category Selection	16	4 (IOB FF)	27	5.759 ns (Max. delay)	5.2	–
*DC base code memory	4	7	7	3.615 ns (Max. delay)	2	–
DCT coeff. Code	15	–	26	9.74 ns (Max. delay)	29.34	–
Run	29	32	55	307	4.13	–
Address Formation	6	–	10	8.40 ns (Max. delay)	22.5	–
AC base code memory	–	–	–	–	3.14	2
**Control	366	229	668	177.17	13.9	–
Complete design	439	273	802	149.535	29.13	2

$ Total power in each case = Dynamic power + 103.13 (Quiescent power)
*****Distributed memory (SRAM) in LUTS have been used for DC base code storage
** 201 Flip Flops have been used for buffer register

implementation and TABLE 4.8 shows the hardware utilization and power consumption report in Xilinx XC2VP30 FPGA device module wise. Here, total power is sum of dynamic power and quiescent power which is 103.13 mW for this device. Also, DC base code module uses SRAMs of LUTs as distributed ROM to store the DC base code table. In this particular implementation, the buffer register width is taken as 201 bits.

Comparison

We have compared our proposed implementation with the existing one and TABLE 4.9 and TABLE 4.10 show the comparison results in terms of memory uses. Agostini *et al.* [19] use 176x21 bits in AC code table and 12x13 bits in DC code table. In our method,

only 161x11 bits in AC and 12x7 bits in DC code table have been used which saves a memory storage of 1997 bits. Sun *et al.*[71] have used strategy to save the memory in code table implementation. They have used 8 bits per AC code table entry. In our proposed strategy, only 7 bits per entry have been used. Thus, the proposed Huffman design is efficient and simple.

TABLE 4.9
COMPARISON OF PROPOSED HUFFMAN CODING IMPLEMENTATION
IN TERMS OF TOTAL MEMORY USES IN TABLE STORAGE

Memory uses	In DC code table	In AC code table
Agostini *et al.* [19]	12x13 bits	176x21 bits
Proposed	12x7 bits	161x11 bits

TABLE 4.10
COMPARISON OF PROPOSED HUFFMAN CODING IMPLEMENTATION
IN TERMS OF NO. OF BITS PER AC TABLE ENTRIES

Memory uses	No. of bits in each AC table entry in 161 locations
Sun *et al.* [71]	8 bits per entry
Proposed	7 bits per entry

4.5 Conclusions

Simulation results for two quantization tables, normal and modified (for hardware simplification), confirm that normalization table that is suitable for hardware simplification can be used in JPEG baseline image compression. A simple FSM based architecture for the computation of 2-D DCT, quantization and zig-zag ordering for JPEG image compression is proposed using quantization matrix suitable for hardware design. It eliminates the 64x13 bits memory requirement for storing the 2-D DCT coefficients as well as another memory requirements for storing the quantized DCT coefficients for zig-zag ordering. Hardware output obtained from FPGA is compared with MATLAB simulation to check the correctness of the implemented design. Huffman coding is implemented using Huffman code table. By employing the strategies, memory requirements to store the AC and DC Huffman code table have been reduced.

Chapter 5

Direct Computation of 8x8 2-D DCT Coefficients Equation and Its Hardware Architecture

5.1 Introduction

The 8x8 2-D DCT is used in image compression algorithms (JPEG) as well as in video compression standard such as MPEG-x and H.26x. It is a highly complex algorithm and its hardware implementation requires a large number of adders and multipliers. Hardware reduction has been the active area of research from the long time. To reduce the hardware, row-column decomposition is the conventional approach to implement 2-D DCT where 1-D DCT is taken to rows followed by 1-D DCT to columns with intermediate results stored in transposition memory. A further hardware saving is obtained by using distributed arithmetic (DA) algorithm which replaces multiplication with additions. The disadvantage of row-column approach is that it requires a large transposition memory to store intermediate 1-D DCT results. This results in high circuit cost, more power consumption and reduced accuracy as final results are obtained after two times DA precision have been applied (first time for 1-D DCT computation and second time for 2-D DCT computation from 1-D DCT). To increase the precision, all intermediate operations need to be done with higher bit-width causing the proportionate increase in size of adders and transposition memory. For low cost consumer products and portable devices; a more regular and simpler circuit is required that has low area and low power dissipation. Row-column decomposition technique is also unsuitable for the applications requiring transmission in limited bandwidth as all DCT coefficients need to be calculated in advance for sending, though it is sent one by one. To reduce the circuit cost, direct recursive computation of 2-D DCT is done using recursive kernel in which

DCT coefficients are computed one by one at regular clock cycles. The disadvantage of recursive kernel is that accuracy is reduced to a large extent due to round-off error. Errors are introduced with the increase of recursive cycles because each processed register values requires higher number of bits for its representation. Fixed size of register in VLSI makes it non-practical.

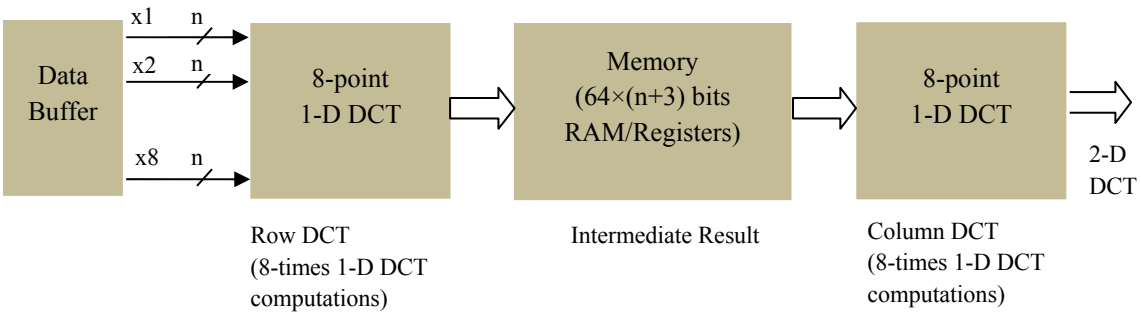
Therefore, non-recursive computation of DCT coefficients which has hardware overhead as low as recursive kernel or even lower is a better choice in VLSI architecture design for the realization of DCT computation. In this chapter, we have proposed non-recursive VLSI architecture for 8x8 2-D DCT that performs direct computation of 2-D DCT without any transposition memory with the following additional advantages.

1) Fractional value multiplication is used two times at last stage only and all intermediate stages are free of error.
2) To increase accuracy, only one register bit-width and one multiplier bit-width need to be changed (almost negligible hardware overhead).
3) Critical/Important DCT coefficients can be calculated first instead of calculating entire 64 DCT coefficients for transmitting in limited bandwidth capacity channel.

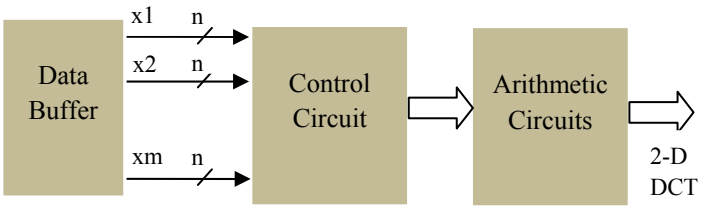
By using this architecture, JPEG image compression has been implemented which requires only one additional register, one multiplier and Huffman coding circuitry bypassing intermediate memory stages to store DCT coefficients.

5.2 Equation for Direct computation of 2-D DCT

Direct computation of 2-D DCT coefficients has advantages that it can be implemented in hardware without transposition memory and most important DCT coefficients can be computed first without computing all 2-D DCT coefficients. Moreover, in row-column approach, 1-D DCT circuitry uses the 16 times DCT computations for 8x8 data (8 times for row DCT and 8 times for column DCT) as shown in Fig. 5.1(a). In terms of hardware requirements, direct computation can be performed by using control and arithmetic circuitry (Fig. 5.1(b)). The design of control circuitry for direct computation plays an important role as it affects the arithmetic circuit requirements and also the accuracy of final coefficients obtained. Here, we derive a novel approach to compute the 2-D DCT



(a)



(b)

Fig. 5.1 2-D DCT computation (a) using 1-D DCT and transposition memory and (b) without transposition memory

coefficients in direct method having a simple control circuitry and less arithmetic circuit units.

The 8x8 2-D DCT for a set of 2-D data $x(i,j)$ with $0 \le i \le 7$ and $0 \le j \le 7$ is given by,

$$F(u,v) = \frac{2}{8} C(u) C(v) \sum_{i=0}^{7} \sum_{j=0}^{7} x(i,j) \times \cos\left(\frac{(2i+1)u\pi}{16}\right) \cos\left(\frac{(2j+1)v\pi}{16}\right) \quad (5.1)$$

where $u, v = 0,1,\ldots,7$, and $C(u), C(v) = \sqrt{1/2}$ for $u, v = 0$ and $C(u), C(v) = 1$, otherwise. For the direct computation of 2-D DCT, recursive computation is the preferred choice. But, hardware implementation of recursive algorithm requires higher number of precisions in the datapath that leads to proportionate increment of area and cost of the design. Therefore, non-recursive algorithm to compute 2-D DCT is better suitable for the dedicated hardware design. Here, we develop the hardware implementation model for the direct computation of 2-D DCT in non-recursive way. DC coefficient $F(0,0)$ is,

$$F(0,0) = \frac{2}{8}C(0)C(0)\sum_{i=0}^{7}\sum_{j=0}^{7}X(i,j)$$
$$= \frac{2}{8}\left(\sqrt{\frac{1}{2}}\right)\left(\sqrt{\frac{1}{2}}\right)\sum_{i=0}^{7}\sum_{j=0}^{7}X(i,j)$$
$$= \frac{1}{8}\sum_{i=0}^{7}\left(X(i,0)+X(i,1)+X(i,2)+X(i,3)+X(i,4)+X(i,5)+X(i,6)+X(i,7)\right)$$

Let,

$$\sum_{i=0}^{7}X(i,0) = sumc1, \quad \sum_{i=0}^{7}X(i,1) = sumc2,$$
$$\sum_{i=0}^{7}X(i,2) = sumc3, \quad \sum_{i=0}^{7}X(i,3) = sumc4,$$
$$\sum_{i=0}^{7}X(i,4) = sumc5, \quad \sum_{i=0}^{7}X(i,5) = sumc6,$$
$$\sum_{i=0}^{7}X(i,6) = sumc7 \text{ and } \sum_{i=0}^{7}X(i,7) = sumc8 \tag{5.2}$$

where, sumc*i* means sum of values in column *i*. Therefore DC coefficient $F(0,0)$, from (5.2), can be written as,

$$F(0,0) = \frac{1}{8}\left(sumc1+sumc2+sumc3+sumc4+sumc5+sumc6+sumc7+sumc8\right) \tag{5.3}$$

$F(1,1)$ is written as,

$$F(1,1) = \frac{2}{8}\sum_{i=0}^{7}\left(\cos\left(\frac{(2i+1)\pi}{16}\right)\right)\times$$
$$\left(\begin{array}{l}\left(X(i,0)-X(i,7)\right)\cos\left(\frac{\pi}{16}\right)+\left(X(i,1)-X(i,6)\right)\cos\left(\frac{3\pi}{16}\right)\\ +\left(X(i,2)-X(i,5)\right)\cos\left(\frac{5\pi}{16}\right)+\left(X(i,3)-X(i,4)\right)\cos\left(\frac{7\pi}{16}\right)\end{array}\right) \tag{5.4}$$

Here, trigonometric property, $\cos(\pi-\varphi) = -\cos(\varphi)$ has been used in (5.1). Equation (5.4) can be further written as,

$$F(1,1)=\left(A\cos\left(\frac{\pi}{16}\right)+(B+E)\cos\left(\frac{3\pi}{16}\right)+(C+I)\cos\left(\frac{5\pi}{16}\right)+(D+M)\cos\left(\frac{7\pi}{16}\right)\right)\times\cos\left(\frac{\pi}{16}\right) \tag{5.5}$$

where, symbols A, B, C, ... represents the values given in TABLE 5.1 in terms of inputs.

TABLE 5.1
SHORT NOTATIONS OF IMAGE DATA VALUES

Image data values				notations n1=(c1-c2) -(c3-c4) n2=(c1-c2)+(c3-c4) n3=(c1+c2)-(c3+c4) n4=(c1+c2)+(c3+c4)			
c1	c2	c3	c4	n1	n2	n3	n4
X(0,0)	X(0,7)	X(7,0)	X(7,7)	A	Ap	pA	pAp
X(1,0)	X(1,7)	X(6,0)	X(6,7)	B	Bp	pB	pBp
X(2,0)	X(2,7)	X(5,0)	X(5,7)	C	Cp	pC	pCp
X(3,0)	X(3,7)	X(4,0)	X(4,7)	D	Dp	pD	pDp
X(0,1)	X(0,6)	X(7,1)	X(7,6)	E	Ep	pE	pEp
X(1,1)	X(1,6)	X(6,1)	X(6,6)	F	Fp	pF	pFp
X(2,1)	X(2,6)	X(5,1)	X(5,6)	G	Gp	pG	pGp
X(3,1)	X(3,6)	X(4,1)	X(4,6)	H	Hp	pH	pHp
X(0,2)	X(0,5)	X(7,2)	X(7,5)	I	Ip	pI	pIp
X(1,2)	X(1,5)	X(6,2)	X(6,5)	J	Jp	pJ	pJp
X(2,2)	X(2,5)	X(5,2)	X(5,5)	K	Kp	pK	pKp
X(3,2)	X(3,5)	X(4,2)	X(4,5)	L	Lp	pL	pLp
X(0,3)	X(0,4)	X(7,3)	X(7,4)	M	Mp	pM	pMp
X(1,3)	X(1,4)	X(6,3)	X(6,4)	N	Np	pN	pNp
X(2,3)	X(2,4)	X(5,3)	X(5,4)	O	Op	pO	pOp
X(3,3)	X(3,4)	X(4,3)	X(4,4)	P	Pp	pP	pPp

Further, the trigonometric property given by,

$$\cos m\times\cos n=\frac{1}{2}\{\cos(m-n)+\cos(m+n)\}$$

can be used in (5.4) to simplify the computation. Therefore, (5.5) becomes,

$$F(1,1)=\frac{1}{8}\left[\begin{array}{l}(A+F+K+P)\cos(0)+(B+E+C+I+H+N-L-O)\cos\left(\frac{\pi}{4}\right)+\\(A+B+E+G+J+L+O-P)\cos\left(\frac{\pi}{8}\right)+(C+I+D+M+F-H-N-K)\cos\left(\frac{3\pi}{8}\right)\end{array}\right]$$

$$= \frac{1}{8}\left[(ACC1)\cos(0) + (ACC2)\cos\left(\frac{\pi}{4}\right) + (ACC3)\cos\left(\frac{\pi}{8}\right) + (ACC4)\cos\left(\frac{3\pi}{8}\right) \right] \quad (5.6)$$

where, ACCi's in (5.6) are used to represent the accumulated input signal values. Following the procedure above, other DCT coefficients can be computed one by one in serial fashion or in parallel according to the requirements by the non-recursive equation which is given as,

$$F(u,v) = R\left[\begin{array}{l} ACC1 \times \left(\frac{1}{8}\cos(\beta 1) \right) + ACC2 \times \left(\frac{1}{8}\cos(\beta 2) \right) + \\ ACC3 \times \left(\frac{1}{8}\cos(\beta 3) \right) + ACC4 \times \left(\frac{1}{8}\cos(\beta 4) \right) \end{array} \right] \quad (5.7)$$

where, $\beta 1$, $\beta 2$, $\beta 3$ and $\beta 4$ represents the cosine angles. $ACC1$, $ACC2$, $ACC3$, $ACC4$ and R values are required to compute DCT coefficients which are listed in TABLE 5.3 whereas TABLE 5.2 lists the short notations used in TABLE 5.3. Angle Group column in TABLE 5.3 represents the four cosine angles $\beta 1$, $\beta 2$, $\beta 3$ and $\beta 4$ whose values are listed in TABLE 5.4. Let us take an example to calculate *F*(3,1). From generalized equation (5.7) and with the help of TABLE 5.3 and 5.4, *F*(3,1) can be written as,

$$F(3,1) = 1 \times \left[\begin{array}{l} S7 \times \left(\frac{1}{8}\cos(0) \right) + (S8 + S9) \times \left(\frac{1}{8}\cos(\pi/4) \right) + \\ (S10 + S11) \times \left(\frac{1}{8}\cos(\pi/8) \right) + (S12 + S13) \times \left(\frac{1}{8}\cos(3\pi/8) \right) \end{array} \right]$$

For the 'Angle Group 3' case, one extra value "c" is added along with four accumulators. From (5.7), it is evident that the required DCT coefficients can be calculated with very high precision as all internal values are preserved accurately because of non-floating type operations involved. Also, to further increase the accuracy, only one register which contains cosine values in fractional format for the multiplication and one multiplier bit-width need to be changed. The DCT coefficients can be calculated in any order by exploiting pipelining and parallelism with throughput and hardware cost trade-off.

TABLE 5.2
SHORT NOTATIONS OF TERMS

Term	notation	Term	notation	Term	notation
$A+F+K+P$	S0	$pA-pM-pC+pO$	S36	$pLp-pNp+pOp-pIp$	S71
$A+B+E+G$	S1	$pJ-pH+pL-pF$	S37	$pAp-pBp-pCp+pDp$	S72
$J+L+O-P$	S2	$pO-pD+pP-pC$	S38	$pNp+pOp-pPp-pMp$	S73
$B+E+C+I$	S3	$pE-pI+pF-pJ$	S39	$pEp-pFp-pGp+pHp$	S74
$H+N-L-O$	S4	$pA-pM-pB+pN$	S40	$pJp+pKp-pLp-pIp$	S75
$C+I+D+M$	S5	$pK+pH-pL-pG$	S41	$pAp-pDp-pEp+pHp$	S76
$F-H-N-K$	S6	$pB-pN-pD+pP$	S42	$pLp+pMp-pPp-pIp$	S77
$E-L-N-C$	S7	$pI-pG+pK-pE$	S43	$pBp-pCp-pFp+pGp$	S78
$A-G-D-K$	S8	$pB-pF-pJ+pN$	S44	$pKp+pNp-pOp-pJp$	S79
$M+J+P-F$	S9	$pC-pG-pK+pO$	S45	$pCp+pNp-pOp-pBp$	S80
$A-G-C-H$	S10	$pA-pE-pI+pM$	S46	$pEp-pHp-pIp+pLp$	S81
$I-J-P+N$	S11	$pD-pH-pL+pP$	S47	$pAp-pDp-pFp+pGp$	S82
$B+O+D+K$	S12	$Ap-Dp+Ep-Hp$	S48	$pJp-pKp-pMp+pPp$	S83
$E+F-M+L$	S13	$Jp-Kp+Np-Op$	S49	$pCp+pNp-pOp-pBp$	S84
$H+I+O-B$	S14	$Ap-Dp+Fp-Gp$	S50	$pEp-pHp-pIp+pLp$	S85
$A-J+D-F$	S15	$Ip-Lp-Np+Op$	S51	$pAp-pDp-pEp+pHp$	S86
$G+P-K-M$	S16	$Bp-Cp+Ep-Hp$	S52	$pLp+pMp-pPp-pIp$	S87
$D-F+E-B$	S17	$Kp+Mp-Pp-Jp$	S53	$pBp-pCp-pFp+pGp$	S88
$L+K+M-O$	S18	$Bp-Cp-Fp+Gp$	S54	$pKp+pNp-pOp-pJp$	S89
$A-J+C-N$	S19	$Ip-Lp-Mp+Pp$	S55	$pA+pE+pI+pM$	S90
$H-G-P-I$	S20	$Ep-Fp-Gp+Hp$	S56	$pB+pF+pJ+pN$	S91
$G-J+M-D$	S21	$Ip-Jp-Kp+Lp$	S57	$pC+pG+pK+pO$	S92
$C-H-B-L$	S22	$Ap-Bp-Cp+Dp$	S58	$pD+pH+pL+pP$	S93
$E+O-I+N$	S23	$Mp-Np-Op+Pp$	S59	$pAp+pEp+pIp+pMp$	S94
$C-H-F+K$	S24	$Ap-Dp-Fp+Gp$	S60	$pBp+pFp+pJp+pNp$	S95
$I-N-D-M$	S25	$Lp-Np+Op-Ip$	S61	$pCp+pGp+pKp+pOp$	S96
$A-P-B-L$	S26	$Ap-Dp-Ep+Hp$	S62	$pDp+pHp+pLp+pPp$	S97
$G+J-E-O$	S27	$Kp+Np-Op-Jp$	S63	sumc1 – sumc8	c1mc8
$pA-pM+pB-pN$	S28	$Bp-Cp+Fp-Gp$	S64	sumc2 – sumc7	c2mc7
$pG-pK+pH-pL$	S29	$Lp-Mp+Pp-Ip$	S65	sumc3 – sumc6	c3mc6
$pA-pM+pC-pO$	S30	$Cp+Ep-Hp-Bp$	S66	sumc4 – sumc5	c4mc5
$pF-pJ-pH+pL$	S31	$Kp-Mp+Pp-Jp$	S67	sumc1 + sumc8	c1pc8
$pB-pN+pD-pP$	S32	$pAp-pDp-pMp+pPp$	S68	sumc2 + sumc7	c2pc7
$pE-pI-pG+pK$	S33	$pFp-pGp-pJp+pKp$	S69	sumc3 + sumc6	c3pc6
$pC-pO-pD+pP$	S34	$pBp-pCp+pEp-pHp$	S70	sumc4 + sumc5	c4pc5
$pE-pI-pF+pJ$	S35				

TABLE 5.3
ACCUMULATOR VALUES AND COSINE ANGLES REQUIRED IN GENERALISED EQUATION (5.7) FOR ALL AC COEFFICIENTS CALCULATION

AC Coeff.	R	$ACC1$	$ACC2$	$ACC3$	$ACC4$	Angle Group	AC Coeff.	R	$ACC1$	$ACC2$	$ACC3$	$ACC4$	Angle Group
$F(1,0)$	$\sqrt{2}$	S90	S91	S92	S93	2	$F(1,4)$	1	S44 + S45	S46 + S47	S46 − S47	S44 − S45	2
$F(2,0)$	$\sqrt{2}$	S94	−S97	S95	−S96	4	$F(2,4)$	1	S86 + S87	S88 + S89	S86 + S87	−(S88 + S89)	4
$F(3,0)$	$\sqrt{2}$	−S92	S90	−S93	−S91	2	$F(3,4)$	1	S46 − S47	−(S44 + S45)	S44 − S45	S46 + S47	2
$F(4,0)$	$\sqrt{2}$	S94	−S95	−S96	S97	3, c=0	$F(4,4)$	1	S72 − S73	S75 − S74	0	0	divide by 8
$F(5,0)$	$\sqrt{2}$	−S91	S93	S90	S92	2	$F(5,4)$	1	S46 + S47	−(S44 − S45)	−(S44 + S45)	−(S46 − S47)	2
$F(6,0)$	$\sqrt{2}$	−S95	S96	S94	−S97	4	$F(6,4)$	1	S76 + S77	−(S78 + S79)	−(S76 + S77)	−(S78 + S79)	4
$F(7,0)$	$\sqrt{2}$	−S93	S92	−S91	S90	2	$F(7,4)$	1	−(S44 − S45)	S46 − S47	−(S46 + S47)	S44 + S45	2
$F(0,1)$	$\sqrt{2}$	c1mc8	c2mc7	c3mc6	c4mc5	2	$F(0,5)$	$\sqrt{2}$	−c2mc7	c4mc5	c1mc8	c3mc6	2
$F(1,1)$	1	S0	S3 + S4	S1 + S2	S5 + S6	1	$F(1,5)$	1	−S7	S8 + S9	−(S12 + S13)	S10 + S11	1
$F(2,1)$	1	S48 + S49	S50 + S51	S52 + S53	S54 + S55	2	$F(2,5)$	1	−(S66 + S67)	S62 + S63	−(S64 + S65)	S60 + S61	2
$F(3,1)$	1	S7	S8 + S9	S10 + S11	S13 − S12	1	$F(3,5)$	1	S21	−(S22 + S23)	S26 + S27	−(S24 + 25)	1
$F(4,1)$	1	S56 + S57	S58 + S59	S58 − S59	S56 − S57	2	$F(4,5)$	1	S58 + S59	−(S56 − S57)	−(S56 + S57)	−(S58 − S59)	2
$F(5,1)$	1	S14	S15 + S16	S17 + S18	S19 + s20	1	$F(5,5)$	1	S0	−(S3 + S4)	S5 + S6	−(S1 + S2)	1
$F(6,1)$	1	−(S64 + S65)	S66 + S67	S60 + S61	S62 + S63	2	$F(6,5)$	1	S50 + S51	−(S54 + S55)	−(S48 + S49)	−(S52 + S53)	2
$F(7,1)$	1	S21	S22 + S23	S24 + S24	S26 + S27	1	$F(7,5)$	1	S14	−(S15 + S16)	S19 + S20	−(S17 + S18)	1
$F(0,2)$	$\sqrt{2}$	c1pc8	−c4pc5	c2pc7	c3pc6	4	$F(0,6)$	$\sqrt{2}$	−c2pc7	c3pc6	c1pc8	−c4pc5	4
$F(1,2)$	1	S28 + S29	S30 + S31	S32 + S33	S34 + S35	2	$F(1,6)$	1	−(S38 + S39)	S42 + S43	S36 + S37	S40 + S41	2
$F(2,2)$	1	S68 − S69	S70 + S71	0	0	3, c= S68+S69	$F(2,6)$	1	S82 + S83	S84 − S85	0	0	3,c= S84+S85
$F(3,2)$	1	S36 + S37	S38 + S39	S40 + S41	S42 + S43	2	$F(3,6)$	1	−(S32 + S33)	S28 + S29	−(S34 + S35)	−(S30 + S31)	2
$F(4,2)$	1	S72 + S73	S74 + S75	S72 + S73	−(S74 + S75)	4	$F(4,6)$	1	−(S74 + S75)	S72 + S73	−(S72 + S73)	−(S74 + S75)	4
$F(5,2)$	1	−(S42 + S43)	S40 + S41	−(S38 + S39)	S36 + S37	2	$F(5,6)$	1	S30 + S31	−(S34 + S35)	−(S28 + S29)	−(S32 + S33)	2
$F(6,2)$	1	S82 + S83	S84 − S85	0	0	3,c= S84+S85	$F(6,6)$	1	S84 − S85	−(S68 − S69)	0	0	3,c= (−S68−S69)
$F(7,2)$	1	S34 + S35	−(S32 + S33)	S30 + S31	−(S28 + S29)	2	$F(7,6)$	1	S40 + S41	−(S36 + S37)	S42 + S43	S38 + S39	2
$F(0,3)$	$\sqrt{2}$	−c3mc6	c1mc8	−c4mc5	−c2mc7	2	$F(0,7)$	$\sqrt{2}$	−c4mc5	c3mc6	−c2mc7	c1mc8	2
$F(1,3)$	1	−S14	S15 + S16	S19 + S20	−(S17 + S18)	1	$F(1,7)$	1	−S21	−(S22 + S23)	S24 + S25	S26 + S27	1
$F(2,3)$	1	S60 + S61	S64 + S65	S62 + S63	S66 + S67	2	$F(2,7)$	1	S54 + S55	−(S52 + S53)	S50 + S51	−(S48 + S49)	2
$F(3,3)$	1	S0	−(S3 + S4)	−(S5 + S6)	S1 + S2	1	$F(3,7)$	1	S14	S15 + S16	−(S17 + S18)	−(S19 + S20)	1
$F(4,3)$	1	S58 − S59	−(S56 + S57)	S56 − S57	S58 + S59	2	$F(4,7)$	1	−(S56 − S57)	S58 − S59	−(S58 + S59)	S56 + S57	2
$F(5,3)$	1	−S21	S22 + S23	S26 + S27	−(S24 + S25)	1	$F(5,7)$	1	−S7	−(S8 + S9)	S10 + S11	S12 + S13	1
$F(6,3)$	1	−(S52 + S53)	S48 + S49	−(S54 + S55)	−(S50 + S51)	2	$F(6,7)$	1	S62 + S63	−(S60 + S61)	S66 + S67	S64 + S65	2
$F(7,3)$	1	−S7	S8 + S9	S12 + S13	−(S10 + S11)	1	$F(7,7)$	1	S0	S3 + S4	−(S1 + S2)	−(S5 + S6)	1
$F(0,4)$	$\sqrt{2}$	c1pc8	−c2pc7	−c3pc6	c4pc5	3,c=0							

TABLE 5.4
ANGLE GROUP VALUES USED IN TABLE 5.3

Angle Group	$(\beta 1, \beta 2, \beta 3, \beta 4)$
1	$(0, \pi/4, \pi/8, 3\pi/8)$
2	$(\pi/16, 3\pi/16, 5\pi/16, 7\pi/16)$
3	$(\pi/4, \pi/4, \pi/4, \pi/4)$
4	$(\pi/8, \pi/8, 3\pi/8, 3\pi/8)$

5.3 Non-recursive VLSI architecture of 2-D DCT

Based on equation (5.7) derived above, the non-recursive architecture for computing 2-D DCT coefficients has the components as shown in Fig. 5.2 along with the clock latency of each components. Since sum of all eight columns are required, it is done by the accumulator (adder). One accumulator can do the accumulation of column values from the data buffer, but it requires 64 clock latency. Therefore, to reduce the clock latency, eight parallel accumulators have been used that gives the sums in eight clock latency.

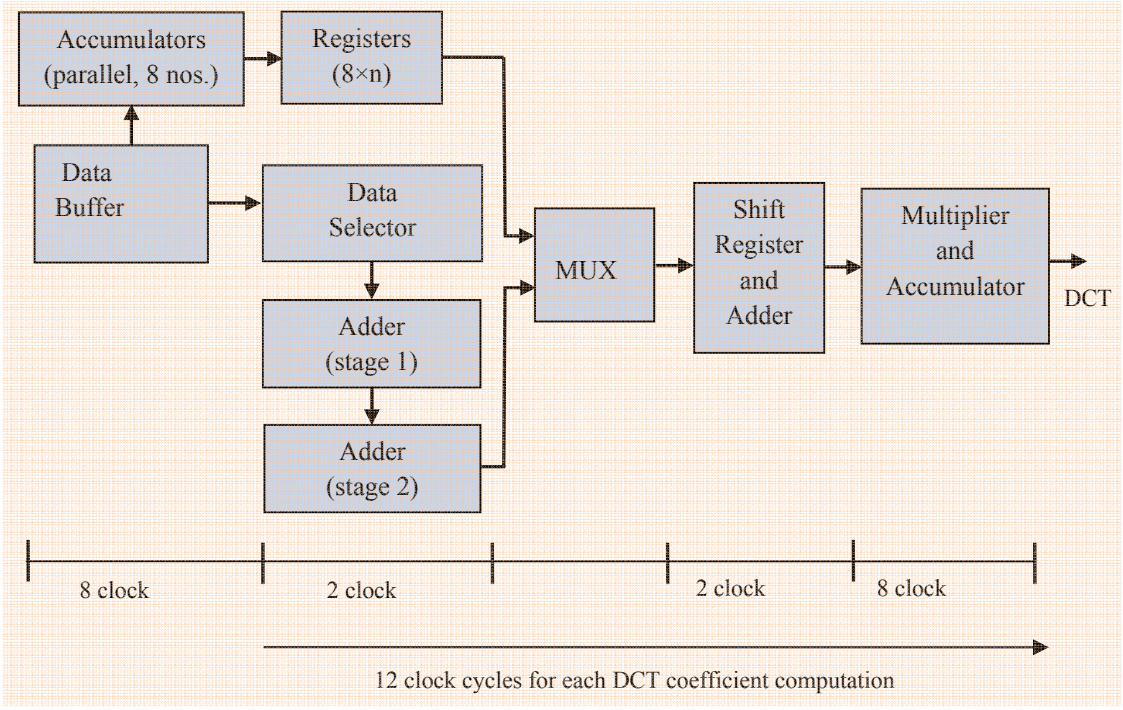


Fig. 5.2 Architectural components of direct 2-D DCT computation

These values are stored in registers of size n-bits. For this design in particular for 8-bits input data, 11-bits registers have been used. Data selector module takes appropriate address of the data inside the data buffer to make available four parallel data at each clock cycle to the adder stage 1. In next clock cycle, four more data comes from the data buffer to the adder stage 1 and the result of successive additions are applied to adder stage 2. So the clock latency for this stage is 2 clock cycles. MUX will select the data either from the column sum registers or from the adder stage 2 and further two successive samples are added in adder and shifter stage which consists of a register and adder as shown in Fig. 5.3. Binary values of cosine in fractional form are stored in one and only register and each ACCi's values are multiplied with a multiplier. Since, the ACCi's values are available after every two clock cycles, four multiplications are performed in 8 clock cycles and results are accumulated. The initial eight clock cycles are used to get the column sum and once it is available, the remaining computations are done in total 12 clock cycles, i.e., successive DCT coefficients are obtained after every 12 clock cycles.

Complete Architecture Design

Fig. 5.4 depicts the one cell component used in proposed architecture of Fig. 5.5. This basic cell adds four data values applied to its four inputs with different sign (positive or negative) and the appropriate sign is selected by multiplexer whose select line is controlled by timing and controller module in Fig. 5.5. From the data buffer, 16 pixels values are obtained in one clock cycle. These 16 pixel values are given to four basic adder modules. The four data sums obtained from the four basic modules are applied further to same type of basic module cell to get addition of these four values.

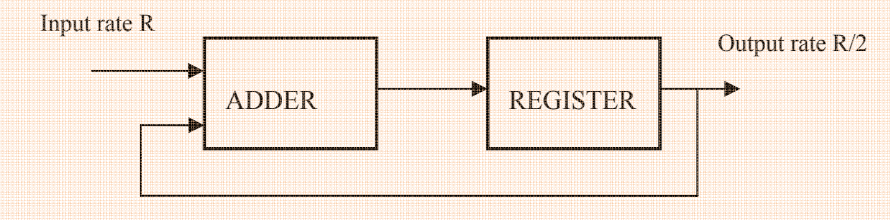


Fig. 5.3 Adder and shifter stage in the architecture

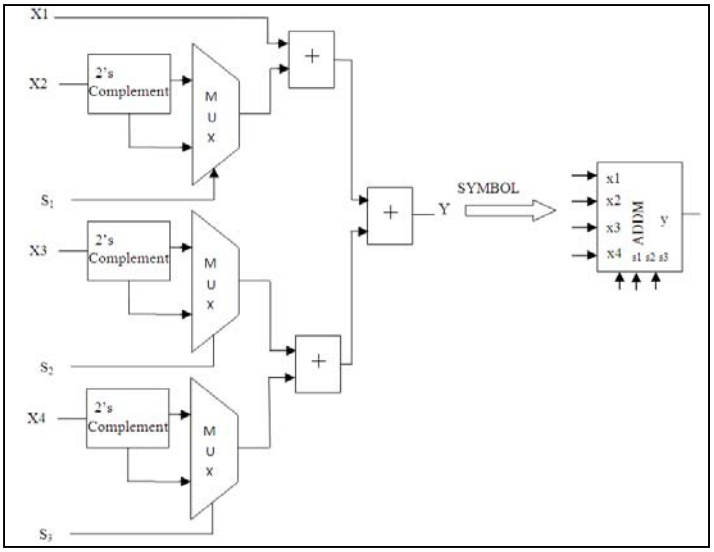


Fig. 5.4 A basic cell to add four inputs with different sign

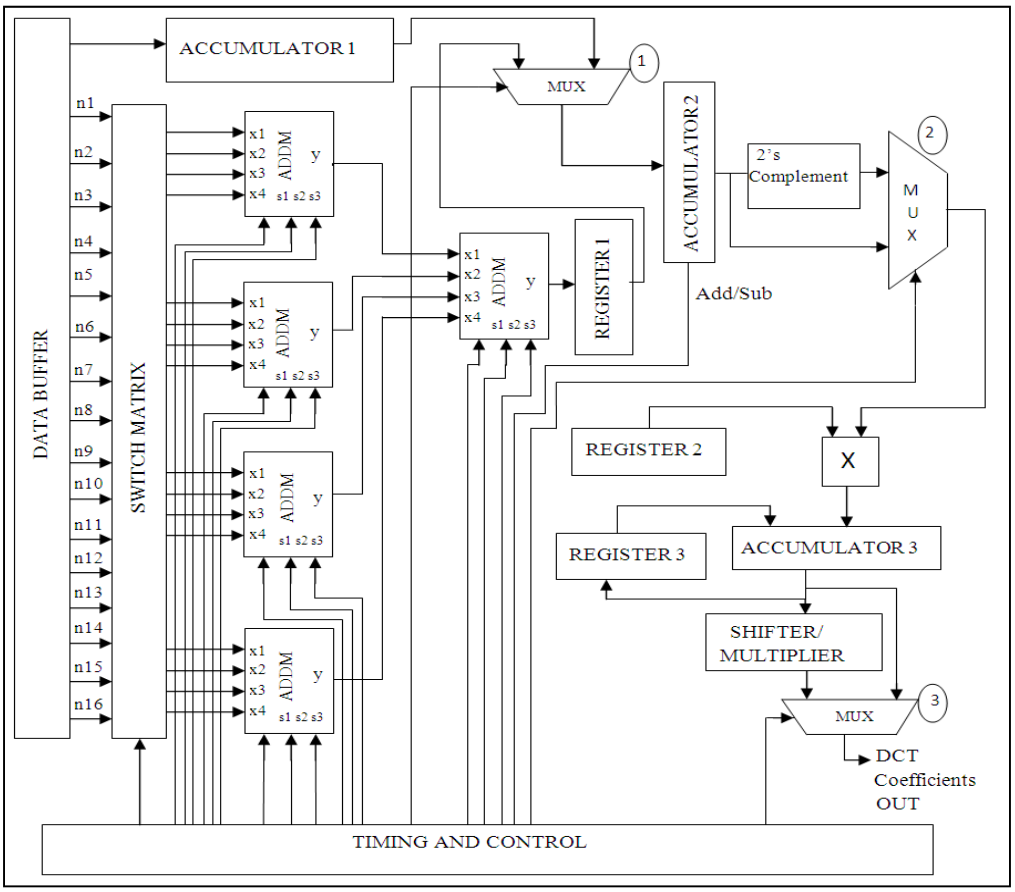


Fig. 5.5 Proposed non-recursive VLSI architecture for the direct computation of 2-D DCT

E.g., let 4 values out of 16 from data buffer are x(0,1), x(0,6), x(7,1) and x(7,6). These values applied to basic adder module with proper sign controlled by timing and control module will result in

$$E = x(0,1)–x(0,6) –x(7,1)+x(7,6)$$

Similarly, others 12 values are given by,

$$L = x(3,2) –x(3,5) –x(4,2)+x(4,5)$$

$$N = x(1,3)–x(1,4) –x(6,3)+x(6,4)$$

and

$$C = x(2,0)–x(2,7) –x(5,0)+x(5,7)$$

These four results when applied to second stage basic adder cell results,

$$S7 = E–L–N–C$$

with proper sign extension.

Other values, e.g., S8, S9, S10, S11 and S12, required for F(3,1) are calculated in next clock cycle. S7 is the ACC1 for F(3,1). ACC2 is sum of S9 and S10. REGISTER 1 is used to hold one value (S8) and next value coming through (S9) is added or subtracted by the ACCUMULATOR 2 with add/sub signal from timing and control circuitry. Sum of columns is calculated by the ACCUMULATOR 1 where a counter is used to get the pixel values from data buffer. Counter output is given as the address in data buffer. MUX 1 is used to select the data from ACCUMULATOR 1 or basic cell module in second stage. MUX 2 is used to change the sign of ACCs as and when required. REGISTER 2 stores the cosine values in binary fractional format (e.g., fractional value of $\cos(\pi/4)$. In every two clock cycle, REGISTER 2 contents are changed and multiplied in the MULTIPLIER. ACCUMULATOR 3 accumulates four values, e.g., S7×(1/8cos(0)), (S8+S9)×(1/8cos(π/4)), (S10+S11)×(1/8cos(π/8)) and (S12+S13)×(1/8cos(3π/8)). SIFTER performs multiplication by √2 where ACCUMULATOR 3 output is shifted and added. SHIFTER reduces the hardware by performing multiplication with less precision by addition of shifted values. Multiplier can be used instead of shifter for higher

precision. Finally MUX 3 selects the values direct from ACCUMULATOR 3 or after one multiplication.

VLSI/FPGA Implementation

The proposed architecture is implemented using 0.18 µm TSMC CMOS standard cell technology library. The industry standard Synopsys Design Compiler (DC) tool has been used for the synthesis of architecture described in VHDL language. 12-bits precision for the cosine values are used in REGISTER 2. For the multiplication by $\sqrt{2}$, shifting operation is performed. TABLE 5.5 shows the comparison of proposed implementation with other existing methods for 2-D DCT computation. The proposed architecture has the lowest area in terms of gate counts (15.4 K) and low power consumption as well (11 mW at 100 MHz).

TABLE 5.5
COMPARISON OF DIFFERENT 2-D DCT ARCHITECTURES

	Shams *et al.* [57]	Chen *et al.* [59]	Chen *et al.* [31]	Sun *et al.* [52]	Chen *et al.* [64]	Jian *et al.* [98]	**Proposed**
Method	Row-column	Row-column	Direct recursive	Row-column	Direct recursive	Direct Non-recursive	**Direct Non-recursive**
Technology	0.18 µm	0.18 µm	NA	2 µm	NA	0.6 µm	**0.18 µm**
ROM words	No	No	No	Yes	No	No	**No**
RAM words	No	No	No	Yes	No	No	**No**
# of Multipliers	0	0	4/6	0	Yes	0	**1**
Gate Counts (NAND2)	22.5 K (approx)	22.2 K	NA	18.25 K (16x16 DCT)	NA	28.5 K	**15.4 K**
Throughput	High	High	Low	Low	Low	Moderate	**Low**
Accuracy	Moderate	Moderate	NA	Very high	Low	NA	**Very high**
Power consumption	Low (0.194 mW for 8x1 DCT)	Moderate (39 mW @125 MHz)	NA	Very high (0.36 W @14.3 MHz)	NA	NA	**Low (11 mW @100 MHz)**
Design complexity	Low	Moderate	High	Low	Moderate	High	**Low**

Note: Gate Count is 1/4 of transistor count as one 2-input NAND gate requires 4 transistors

Layout of the Design

Additional registers (64x8 bits) for the data buffer have been used with 8-bits data input for storing 64 data values in 64 clock cycles. The gate level design mapped in 0.18 μm TSMC library is obtained after synthesis from the DC in verilog form. The layout of the design is performed in Cadence SOC encounter tool. Fig. 5.6 shows the automatic layout generated.

DCT coefficients Calculation in Any order

Although throughput is low, it calculates the DCT coefficients in any order. Hence, it finds more importance in applications like image and video compressions. In the mentioned applications, only few low order DCT coefficients are important (carry much of the visual information) and others are quantized to zero value. In JPEG image compression, quantized DCT coefficients are rearranged in zig-zag order [15]. In terms of hardware implementation of JPEG and MPEG, additional storage (to store 64 DCT coefficients) and reordering circuitry (control circuit) are required as shown in Fig. 5.7(a) [19, 69–71]. Moreover, quantization table which consists of 64 memory locations (ROM) as quantization levels are also used (Fig 5.7(b)). This burden can be overcome by getting

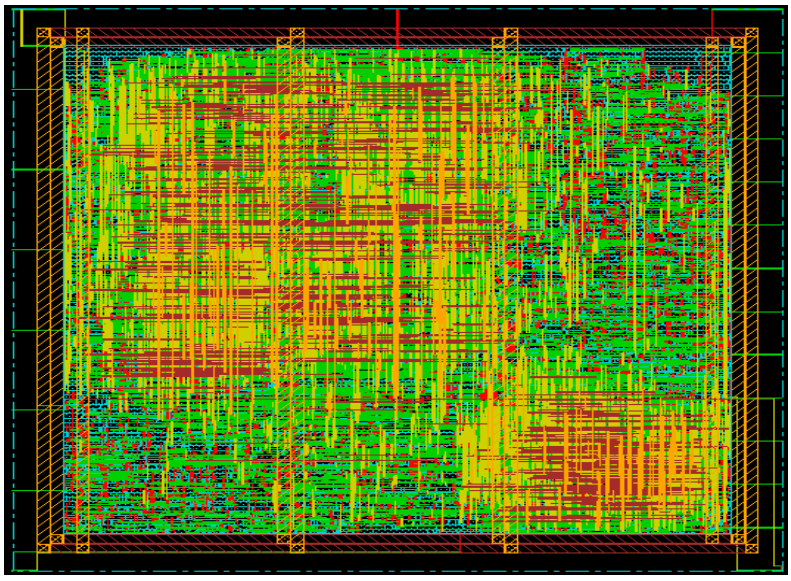
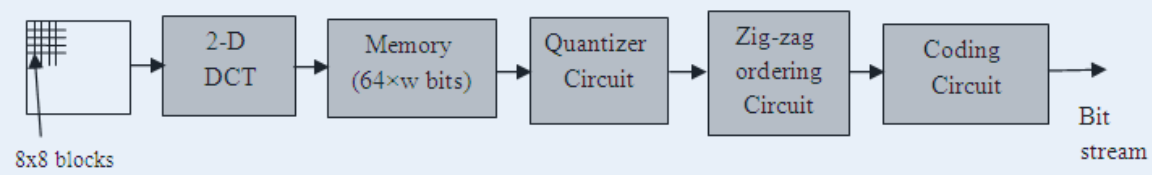
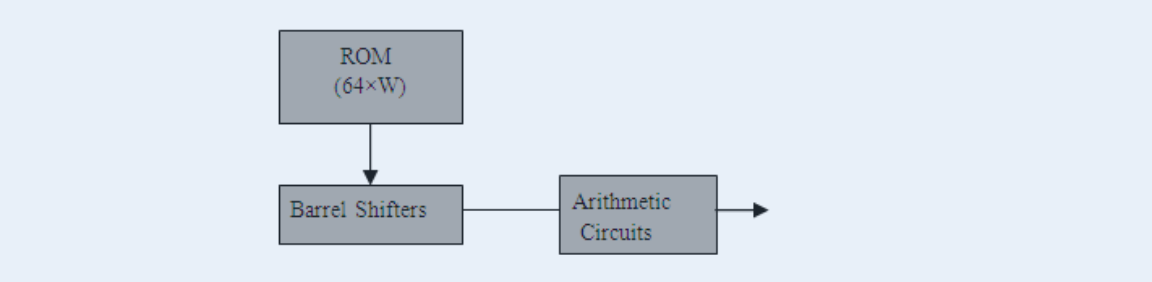


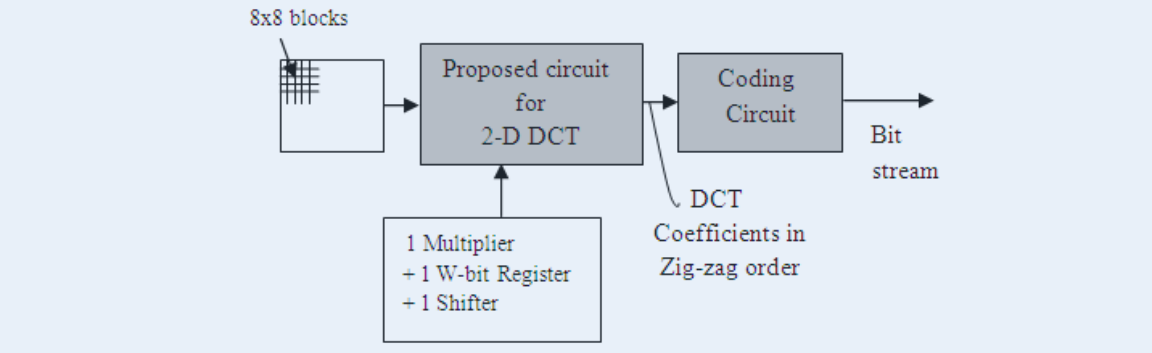
Fig. 5.6 Layout of proposed 2-D DCT design with 64x8 bits registers for the data buffer



(a)



(b)



(c)

Fig. 5.7 JPEG Image and MPEG video compression flow with (a) general DCT model, (b) quantizer circuit and (c) proposed DCT model

zig-zag ordered coefficients directly from the proposed architecture with the control circuitry designed to obtain coefficients in zig-zag order. Also, instead of using 64xW size of ROM for quantization level of coefficient, one register of width W is used and its value is changed by the same control circuitry at the time of getting final DCT output. Since image can be reconstructed with few low order DCT coefficients, low bandwidth can be used for the transmission of images by calculating few low order DCT coefficients with the help of proposed architecture model. Fig. 5.7(c) shows the simple model used in the proposed design.

Low Hardware overhead for higher Precision

In DA based DCT computation, for getting 1-D DCT results with higher accuracy even without having input width unchanged, DA precision needs to be increased. This leads to increase in bit-width of adders and consequently the area of chip[57, 59]. The proposed architecture has all intermediate operations performed in fixed point format. The only fractional values used in the computation are the cosine values stored in a single register and it is multiplied with the accumulated value by a multiplier. Therefore, to increase the accuracy of the DCT coefficients, only one register bit-width and one multiplier input bit-width has to be changed. Thus, a negligible hardware overhead is required for high accuracy.

Simulation and Image Reconstruction

An image consists of huge amount data. Therefore, to simulate the modelled architecture for different images with different precision is time consuming. MATLAB model of the architecture is designed and data samples are compared with the DCT output obtained from the VHDL simulation in Xilinx ISE 10.1 with the MATLAB. The similarity of outputs confirm that MATLAB model of the design is exact. Using the MATLAB model, two standard test images named Lena and Peppers data are DCT transformed. For the reconstruction of the images, Inverse DCT (IDCT) is performed using MATLAB double precision floating point data (Fig. 5.8). The original and reconstructed images are shown in Fig. 5.9. PSNR is the objective quality image metric. It is better suited for objective quality measurement of design modelled in hardware as it

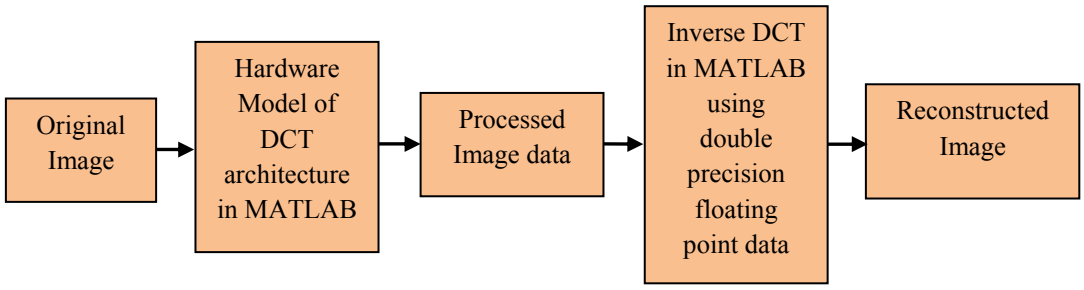


Fig. 5.8 Image processing using proposed 2-D DCT architecture model in MATLAB

(a) (b)

(c) (d)

Fig. 5.9 Original, (a) and (c), and reconstructed, (b) and (d), images using proposed non-recursive 2-D DCT architecture model

focuses on error introduced by the truncation [57]. Fig. 5.10 depicts the PSNR comparison performances for proposed architecture with other existing at different internal bit-width. In the proposed architecture, bit-width of register which stores the cosine floating point data is considered. The proposed architecture has very high accuracy in terms of PSNR as all internal computations are in fixed point. PSNR up to 56 dB has been obtained with the proposed architecture model which was obtained in [52] using 16x16 transform whereas all other transform based on 8x8 has PSNR a maximum up to 50 dB.

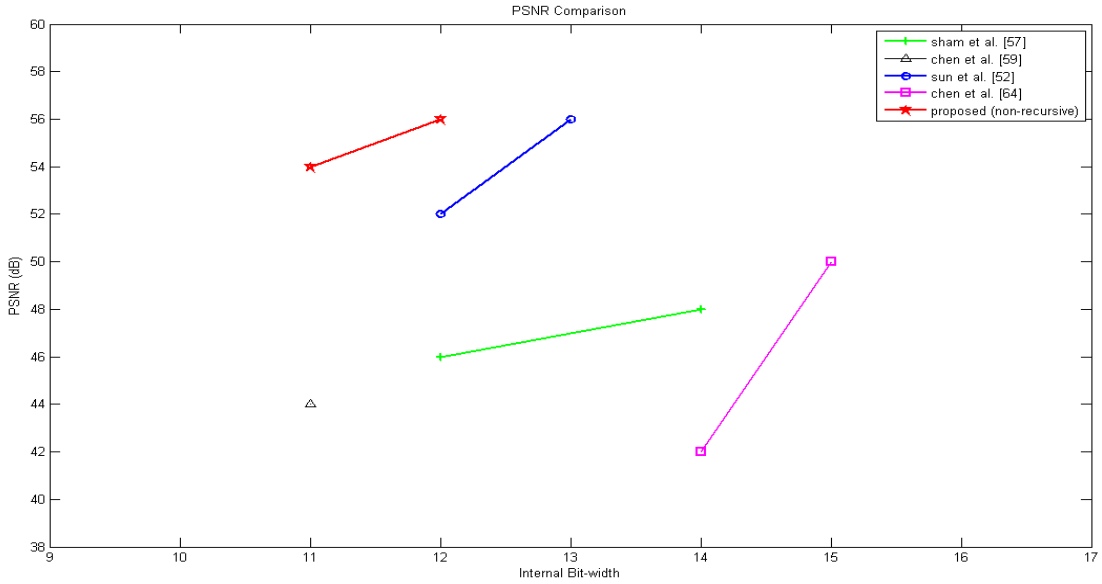


Fig. 5.10 PSNR with different internal bit-width precision used

FPGA prototyping for Silicon validation

The proposed architecture is prototyped in Xilinx FPGA for the silicon validation. TABLE 5.6 shows the hardware resource utilization in XC2VP30 device on Virtex-II pro board and its comparison with other architectures. Since no transposition memory is used, it has low register (379 numbers) utilization. These registers are used to store intermediate calculations and also states of the timing and control circuitry.

TABLE 5.6
FPGA IMPLEMENTATION AND COMPARISON RESULT OF PROPOSED NON-RECURSIVE 2-D DCT ARCHITECTURE WITH DA

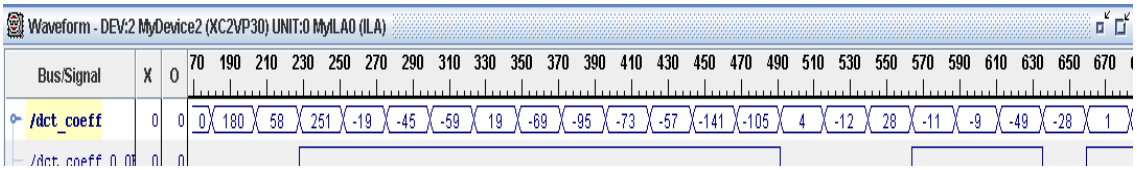
FPGA-chip Xilinx XC2VP30	2-D DCT Architecture implemented using row-column in DA	Proposed non-recursive 2-D DCT Architecture
# of four input LUTs	4502	3370
# of slices	2435	1747
# of Slice Flip Flops	868	379
Max. Frequency (MHz)	48.4	58.6
power consumption	14.97 W	0.79 W

Note: The proposed architecture uses one 18X18 Multiplier

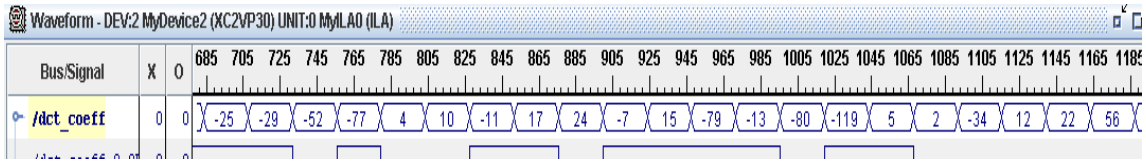
A sample 8x8 data which is given by,

$$Di = \begin{bmatrix} -95 & -96 & 99 & 98 & 94 & 100 & 57 & 54 \\ 108 & 103 & 99 & 98 & 94 & 80 & 57 & 39 \\ 107 & 102 & -2 & -3 & 94 & 80 & 58 & -60 \\ 104 & 99 & 96 & 96 & -7 & -20 & 58 & 41 \\ 2 & -3 & -6 & -5 & -7 & -19 & -41 & -58 \\ -1 & -5 & -8 & -6 & -8 & -19 & 60 & 42 \\ -3 & -7 & -9 & -7 & -8 & -19 & -40 & -57 \\ -4 & -7 & -10 & -8 & -8 & -19 & -40 & -57 \end{bmatrix}$$

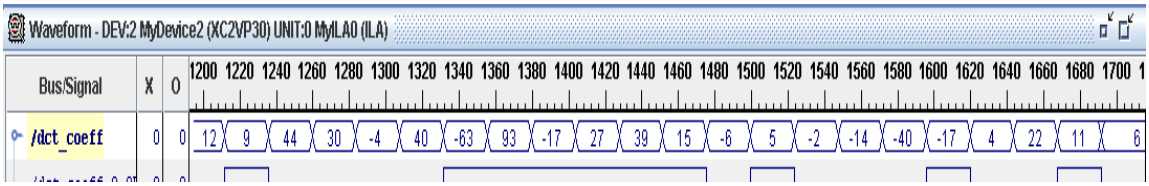
is taken for the functional verification. These data are stored in block RAM of the Xilinx FPGA XC2VP30 device on Virtex-II pro board. The device is programmed by bit-file generated from the ISE tool. The hardware output is obtained from device using ChipScope Pro logic analyser tool through USB cable. Fig. 5.11 shows the output from the device in zig-zag order with 50 MHz clock frequency (digital clock manager is used to get 50 MHz frequency from 100 MHz available on the Virtex-II pro board). The DCT



(a)



(b)



(c)

Figure 5.11 Zig-zag order DCT coefficients (a) 1 to 21, (b) 22 to 42 and (c) 43 to 64 obtained from Xilinx xc2vp30 device using ChipScope pro logic analyser

output from the MATLAB model used for the image transform and reconstruction for PSNR measurement is given by,

$$DCT(Di) = \begin{bmatrix} 180 & 58 & -59 & 19 & -12 & 28 & -11 & 17 \\ 251 & -45 & -69 & 4 & -11 & 10 & 24 & 12 \\ -19 & -92 & -105 & -9 & 4 & -7 & 56 & 8 \\ -73 & -141 & -49 & -77 & 15 & 22 & 44 & 12 \\ -57 & -28 & -52 & -79 & 11 & 30 & 39 & -6 \\ 1 & -29 & -13 & -34 & -4 & 27 & 5 & 4 \\ -25 & -81 & 2 & 43 & -17 & -4 & -17 & 22 \\ -119 & 5 & -63 & 93 & -14 & -40 & 11 & 6 \end{bmatrix}$$

The MATLAB output, same as obtained through the hardware, confirms the silicon validation of the proposed 2-D DCT architecture. The quantized DCT coefficients obtained from the hardware by using JPEG standard quantization level [15] is shown in Fig. 5.12. It is obtained by inserting an additional 16-bits register and a multiplier only in the 2-D DCT circuitry. The quantized output is obtained in next clock cycle from the DCT coefficients because of the use of a register between DCT and multiplier.

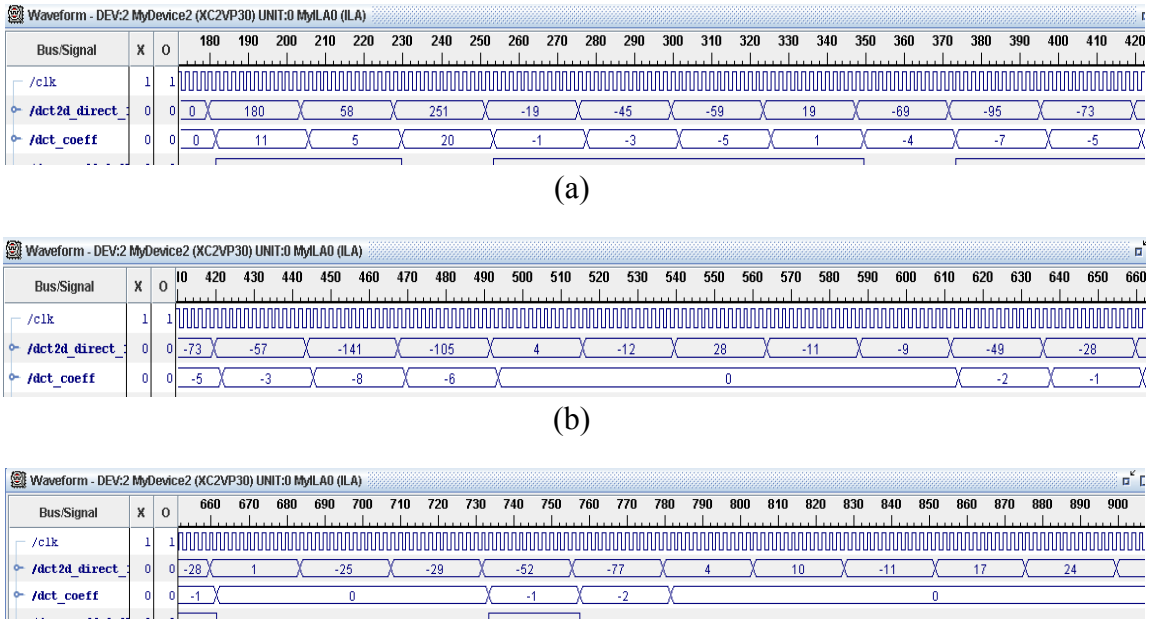
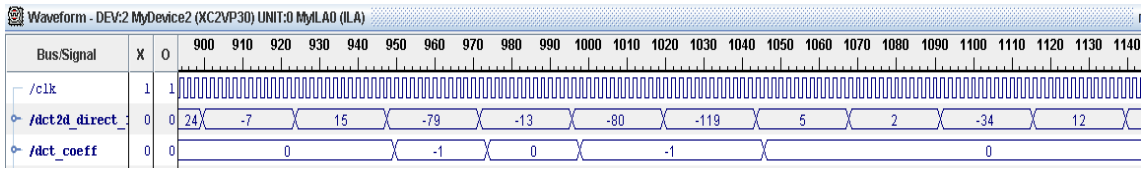
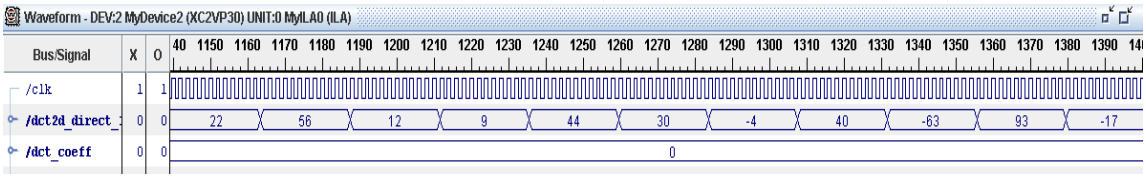


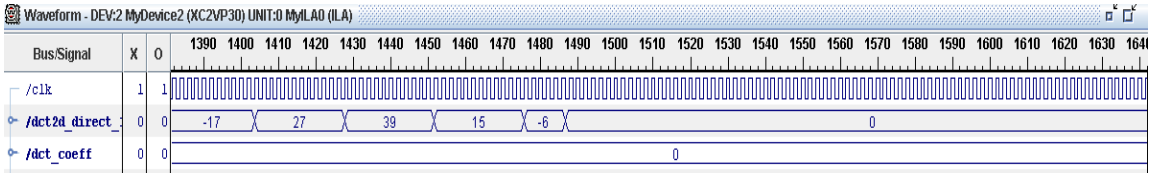
Fig. 5.12 Zig-zag ordered DCT coefficients along with Quantization (a) 1 to 10, (b) 11 to 20, (c) 21 to 30



(d)



(e)



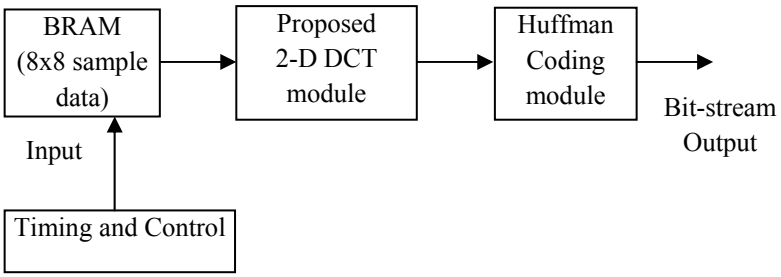
(f)

Fig. 5.12 (continued), (d) 31 to 40, (e) 41 to 51 and (f) 51 to 64 obtained from Xilinx xc2vp30 device using ChipScope pro logic analyzer

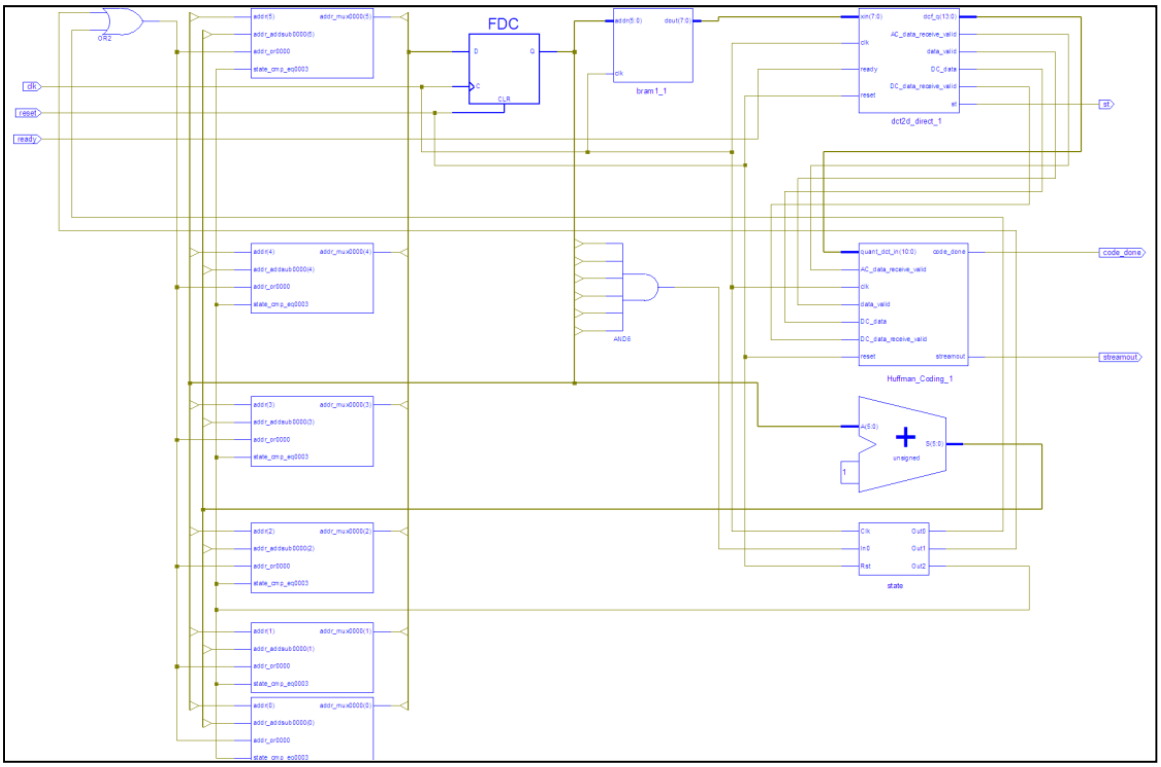
5.4 JPEG Image Compression Architecture using Proposed Non-recursive 2-D DCT

The non-recursive 2-D DCT architecture proposed in the previous section is used for the JPEG image compression architecture. Since, this 2-D DCT architecture gives quantized and zig-zag ordered coefficients, the only other module used here is Huffman coding architecture (explained in previous chapter). The 2-D DCT architecture gives quantized DCT coefficients at 12 clock cycles per coefficient. The developed Huffman coding architecture performs Huffman coding and stores the codes in memory at each clock cycle. For small quantized (DCT) coefficients, base code length will be small along with coefficients code itself. Therefore, for less than 12-bits encoding per coefficient, same clock cycle can drive both the 2-D DCT module and Huffman coding module. However, if Huffman coding length is bigger (more than 12-bits per coefficient value), then clock of Huffman coding can be halved for correct functioning. Here, since coefficients are small, same clock has been used in both the module. The signals required to indicate the valid AC and DC coefficients are generated from the DCT module by inserting those

signals high and low at the appropriate time with the help of 1-bit register for each signal (no additional hardware overhead). Fig. 5.13(a) shows the JPEG compression model using proposed non-recursive 2-D DCT architecture. The complete removal of intermediate stages for storage of DCT coefficients for quantization and zig-zag ordering is illustrated using proposed model. Fig. 5.13(b) shows the RTL schematic of the architecture in Xilinx FPGA. The same sample data used in the DCT and quantization is used here. Fig. 5.14 shows the simulation and hardware outputs from FPGA (same in all



(a)



(b)

Fig. 5.13 Architecture of JPEG compression using proposed non-recursive 2-D DCT (a) block diagram and (b) RTL Schematic in Xilinx

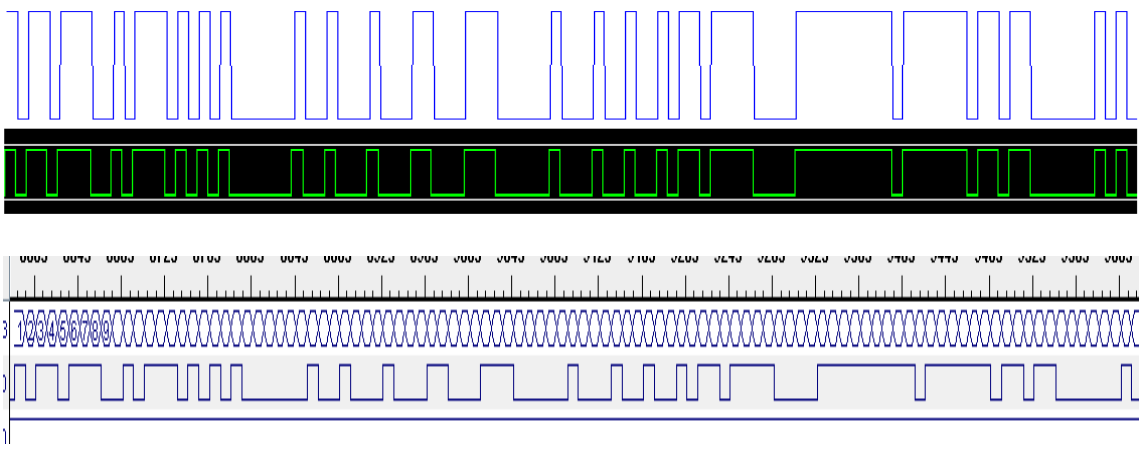


Fig. 5.14 Bit stream of 8x8 sample JPEG processed data using non-recursive 2-D DCT architecture model in MATLAB (top), VHDL simulation (middle) and Hardware output through Xilinx ChipScope Pro (bottom)

three cases). The simulation output from the MATLAB is one that has been used for image reconstruction using proposed non-recursive 2-D DCT architecture model. The clock frequency used here for testing on Xilinx Virtex-II pro board is 10 MHz. TABLE 5.7 shows the hardware utilization summary in FPGA. It can be noted that additional 8-bits 64 buffer has been used for the sample data storage apart from JPEG main circuitry.

TABLE 5.7
FPGA IMPLEMENTATION RESULTS OF COMPLETE JPEG USING PROPOSED NON-RECURSIVE 2-D DCT

FPGA-chip: Xilinx XC2VP30			
	Used	Available	Utilization
# of four input LUTs	4427	27392	16%
# of slices	2325	13696	16%
# of Slice Flip Flops	1137	27392	4%
# of BRAMs	3	136	2%
# of MULT18x18s	2	136	1%
Max. Frequency (MHz)	37.92	–	–
Power consumption	140 mW	–	–

5.5 Conclusions

A novel non-recursive VLSI architecture for the direct computation of 2-D DCT is proposed and implemented in 0.18 μm ASIC library as well as in FPGA. Implementation results show that the proposed architecture is area and power efficient when compared with other architectures which compute 2-D DCT either by direct or row-column method. Although the architecture has low throughput, it has very high accuracy in terms of PSNR. High accuracy is due to preserving all internal calculations in integer format. To further increase the accuracy, hardware overhead is negligibly small. The proposed architecture can calculate DCT coefficients in any order finding it more suitable for image and video compression applications where few low order DCT coefficients are enough for the reconstruction of images. Complete JPEG architecture is implemented using this non-recursive DCT architecture and Huffman coding with complete removal of intermediate storages in different stages like quantization and zig-zag ordering. FPGA prototyping has been done for the silicon validation in Xilinx FPGA. Hardware output is same as obtained from MATLAB and VHDL simulations.

Chapter 6

Summary and Conclusions

6.1 Summary

In the implementation of image compression algorithms, hardware platform provides the faster speed than their software counterpart. Also, there is maximum energy efficiency obtained when the design is implemented in dedicated hardware. For the portable devices running on battery, there is a need to reduce the power as well as silicon area in the hardware circuitry (to reduce the cost). Architecture exploration is one of the optimization steps in VLSI where different architectures are explored to obtain the required specification at the lowest silicon area.

In the proposed work, the image compression algorithms specially DCT, DHT, Huffman coding and JPEG are explored and implemented for the purpose of reduced silicon area and power.

Chapter-1 introduces the topic along with motivation and work done so far in the DCT, DHT and Huffman coding architecture implementations.

Chapter-2 briefs the basics of image compression and image quality metric. The energy compaction property of DCT has been studied with the image compression and decompression using selected DCT coefficients. It is found that first 15 zig-zag ordered DCT coefficients are enough for the image reconstruction while providing extra compression. So, the circuit which computes 2-D DCT coefficients one by one will be a good choice for DCT based image compression. The separable DHT has been used for image compression and decompression in JPEG style and it is found that it performs almost same as DCT at very high compression.

Chapter-3 introduces the efficient hardware implementation algorithm called DA. Both ROM based and ROM free approaches have been used for the DHT implementation. ROM free DA requires less hardware than ROM based DA and has more accuracy. Efficient ROM free 1-D DCT architecture is proposed and implemented using DA approach which has 31% area improvement and 37 % power improvement than the conventional.

Chapter-4 is dedicated towards the removal of intermediate stages which requires a significant amount of ROMs and registers to store DCT coefficients in quantization and zig-zag ordering in JPEG image compression. Also, Huffman coding architecture has been implemented using the strategies to save the memory in the storage of Huffman code tables.

Chapter-5 presents a novel non-recursive architecture for the computation of 2-D DCT coefficients without intermediate transposition memory. The architecture is implemented in FPGA as well as in 0.18 μm ASIC library. The comparative result shows that the architecture is area efficient in terms of gate counts. The architecture has excellent image quality in terms of PSNR. The additional feature of this architecture is that it can compute the 2-D DCT coefficients in any order (zig-zag ordering has been implemented in the current work). Using this architecture, the complete JPEG image compression architecture is implemented. The only additional components required till zig-zag ordering are one 13-bits register and one 13x13 multiplier to perform quantization. Huffman coding is integrated without the buffer in between quantized DCT coefficients and Huffman coding module. The prototyping in Xilinx FPGA of the complete JPEG architecture and comparison of the results obtained from MATLAB and VHDL simulations (same in all cases) validated the designed JPEG.

6.2 General Conclusions

The following are the conclusions from the research carried out:

1) Presented direct non-recursive computation approach is the most suitable design for hardware implementation when high image quality is required at low cost hardware. The quantized and zig-zag ordered coefficients obtained through this

non-recursive architecture completely removes the intermediate stages like memory for storing quantization table and DCT coefficients at different stages resulting in low cost image compression architecture.

2) For the applications requiring only DCT coefficients with high throughput, proposed efficient ROM DA based 1-D DCT circuitry can be used which has low area and low power consumption than the conventional ROM free DA.
3) SDHT can be employed in the system where high compression of image is required at high compression. It has same performance as DCT in terms of PSNR while hardware is significantly lower than DCT.
4) The modified quantization table suitable for hardware simplification has the same performance in terms of PSNR as default one provided by JPEG. However, it has no storage requirement in memory and FSM based design approach leads to memory reduction in storage of DCT coefficients for zig-zag ordering and quantization.
5) The Huffman coding architecture has been implemented with the reduced memory for the storage of Huffman code tables and it encodes the coefficients bit-by-bit at each clock cycle resulting in efficient design.

6.3 Future Scope

The non-recursive equation for the direct computation of 2-D DCT coefficients is very much suitable for the image and video compression architectures design as it computes the 2-D DCT coefficients in any order. The future scope can be to optimize the architecture using this equation for the high throughput image compression using pipeline design technique.

Appendix. A

TABLE A.1
*BASE CODES FOR DC COEFFICIENTS

Category	Codeword
0	00
1	010
2	011
3	100
4	101
5	110
6	1110
7	11110
8	111110
9	1111110
10	11111110
11	111111110

*Source : R. C. Gonzalez, R. E. Woods, Digital Image Processing, 2nd.Ed.,Prentice Hall, 2002.

Appendix. B

TABLE A.2
*BASE CODES FOR AC COEFFICIENTS

(Run, Cat.)	AC Codeword	(Run, Cat.)	AC Codeword	(Run, Cat.)	AC Codeword	(Run, Cat.)	AC Codeword
0,0	1010 (eob)						
0,1	00	4,1	111011	8,1	11111010	12,1	1111111010
0,2	01	4,2	1111111000	8,2	111111111000000	12,2	1111111111011010
0,3	100	4,3	1111111110010111	8,3	1111111110110111	12,3	1111111111011011
0,4	1011	4,4	1111111110011000	8,4	1111111110111000	12,4	1111111111011100
0,5	11010	4,5	1111111110011001	8,5	1111111110111001	12,5	1111111111011101
0,6	111000	4,6	1111111110011010	8,6	1111111110111010	12,6	1111111111011110
0,7	1111000	4,7	1111111110011011	8,7	1111111110111011	12,7	1111111111011111
0,8	1111110110	4,8	1111111110011100	8,8	1111111110111100	12,8	1111111111100000
0,9	1111111110000010	4,9	1111111110011101	8,9	1111111110111101	12,9	1111111111100001
0,10	1111111110000011	4,10	1111111110011110	8,10	1111111110111110	12,10	1111111111100010
1,1	1100	5,1	1111010	9,1	111111000	13,1	11111111010
1,2	111001	5,2	1111111001	9,2	1111111110111111	13,2	1111111111100011
1,3	1111001	5,3	1111111110011111	9,3	1111111111000000	13,3	1111111111100100
1,4	111110110	5,4	1111111110100000	9,4	1111111111000001	13,4	1111111111100101
1,5	11111110110	5,5	1111111110100001	9,5	1111111111000010	13,5	1111111111100110
1,6	1111111110000100	5,6	1111111110100010	9,6	1111111111000011	13,6	1111111111100111
1,7	1111111110000101	5,7	1111111110100011	9,7	1111111111000100	13,7	1111111111101000
1,8	1111111110000110	5,8	1111111110100100	9,8	1111111111000101	13,8	1111111111101001
1,9	1111111110000111	5,9	1111111110100101	9,9	1111111111000110	13,9	1111111111101010
1,10	1111111110001000	5,10	1111111110100110	9,10	1111111111000111	13,10	1111111111101011
2,1	11011	6,1	1111011	10,1	111111001	14,1	111111110110
2,2	11111000	6,2	11111111000	10,2	1111111111001000	14,2	1111111111101100
2,3	1111110111	6,3	1111111110100111	10,3	1111111111001001	14,3	1111111111101101
2,4	1111111110001001	6,4	1111111110101000	10,4	1111111111001010	14,4	1111111111101110
2,5	1111111110001010	6,5	1111111110101001	10,5	1111111111001011	14,5	1111111111101111
2,6	1111111110001011	6,6	1111111110101010	10,6	1111111111001100	14,6	1111111111110000
2,7	1111111110001100	6,7	1111111110101011	10,7	1111111111001101	14,7	1111111111110001
2,8	1111111110001101	6,8	1111111110101100	10,8	1111111111001110	14,8	1111111111110010
2,9	1111111110001110	6,9	1111111110101101	10,9	1111111111001111	14,9	1111111111110011
2,10	1111111110001111	6,10	1111111110101110	10,10	1111111111010000	14,10	1111111111110100
3,1	111010	7,1	11111001	11,1	1111111010	15,1	1111111111110101
3,2	111110111	7,2	11111111001	11,2	1111111111010001	15,2	1111111111110110
3,3	11111110111	7,3	1111111110101111	11,3	1111111111010010	15,3	1111111111110111
3,4	1111111110010000	7,4	1111111110110000	11,4	1111111111010011	15,4	1111111111111000
3,5	1111111110010001	7,5	1111111110110001	11,5	1111111111010100	15,5	1111111111111001
3,6	1111111110010010	7,6	1111111110110010	11,6	1111111111010101	15,6	1111111111111010
3,7	1111111110010011	7,7	1111111110110011	11,7	1111111111010110	15,7	1111111111111011
3,8	1111111110010100	7,8	1111111110110100	11,8	1111111111010111	15,8	1111111111111100
3,9	1111111110010101	7,9	1111111110110101	11,9	1111111111011000	15,9	1111111111111101
3,10	1111111110010110	7,10	1111111110110110	11,10	1111111111011001	15,10	1111111111111110
						15,0	111111110111 (Spl. Code)

*Source : R. C. Gonzalez, R. E. Woods, Digital Image Processing, 2nd.Ed.,Prentice Hall, 2002.

References

[1] R. C. Gonzalez, R. E. Woods, Digital Image Processing, 2nd.Ed.,Prentice Hall, 2002.

[2] Chin-Hwa Kuo, Tzu-Chuan Chou and Tay-Shen Wang, "An efficient spatial prediction-based image compression scheme," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.12(10), pp. 850- 856, Oct. 2002.

[3] Chen Shoushun, Amine Bermak, Wang Yan and Dominique Martinez, "Adaptive-Quantization Digital Image Sensor for Low-Power Image Compression," *IEEE Transactions on Circuits and Systems* I: Regular Papers, vol.54(1), pp.13-25, Jan. 2007.

[4] M.D. Reavy, C.G.Boncelet, "An algorithm for compression of bilevel images," *IEEE Transactions on Image Processing*, vol.10(5), pp.669-676, May 2001.

[5] Debin Zhao, Wen Gao, and Y. K. Chan, "Morphological representation of DCT coefficients for image compression," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.12(9), pp. 819-823, Sep. 2002.

[6] K. A. Kotteri, A. E. Bell and J. E. Carletta, "Multiplierless filter Bank design: structures that improve both hardware and image compression performance," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.16(6), pp. 776- 780, June 2006.

[7] N. N. Ponomarenko, K. O.Egiazarian, V. V.Lukin, and J. T. Astola, "High-Quality DCT-Based Image Compression Using Partition Schemes," *IEEE Signal Processing Letters*, vol.14(2), pp.105-108, Feb. 2007.

[8] Xinpeng Zhan, "Lossy Compression and Iterative Reconstruction for Encrypted Image," *IEEE Transactions on Information Forensics and Security*, vol.6(1), Mar. 2011.

[9] Yi-Huang Han and Jin-Jang Leou, "Detection and correction of transmission errors in JPEG images," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.8(2), pp.221-231, Apr. 1998.

[10] R.Chandramouli, N.Ranganathan and S.J. Ramadoss, "Adaptive quantization and fast error-resilient entropy coding for image transmission," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.8(4), pp.411-421, Aug. 1998.

[11] V. DeBrunner, L. DeBrunner, Wang Longji and S. Radhakrishnan, "Error control and concealment for image transmission," *IEEE Communications Surveys & Tutorials*, vol.3(1), pp.2-9, First Quarter 2000.

[12] P.P. Dang and P.M. Chau, "Robust image transmission over CDMA channels," *IEEE Transactions on Consumer Electronics*, vol.46(3), pp.664-672, Aug 2000.

[13] K. S. Thyagarajan, Digital Image Processing with Application to Digital Cinema, Focal Press, Elsevier, 2006.

[14] Web, http://www.jpeg.org

[15] Gregory K. Wallace, "The JPEG Still Picture Compression Standard," *IEEE Transactions on Consumer Electronics*, Vol. 38(I), Feb. 1992.

[16] M.A. Golner, W.B. Mikhael and V. Krishnan, "Multifidelity JPEG based compression of images using variable quantisation," *Electronics Letters*, vol. 37(7), pp.423-424, March 2001.

[17] Navin Chaddha, Avneesh Agrawal, Anoop Gupta and Teresa H.Y. Meng, "Variable compression using JPEG," *Proceedings of the International Conference on Multimedia Computing and Systems*, 1994., pp.562-569,May 1994.

[18] Long- Wen Chang and Ching- Yang Wang and Shiuh-Ming Lee, "Designing JPEG quantization tables based on human visual system," *Proceedings. 1999 International Conference on Image Processing, ICIP* 99. pp.376-380, 1999.

[19] Luciano Volcan Agostini, Ivan Saraiva Silva and Sergio Bampi, "Multiplierless and fully pipelined JPEG compression soft IP targeting FPGAs," *Microprocessors and Microsystems*, vol. 31(8), 3 pp.487-497, Dec. 2007.

[20] J.A. Kalomiros and J. Lygouras, "Design and evaluation of a hardware/software FPGA-based system for fast image processing," *Microprocessors and Microsystems*, vol.32, pp.95–106, 2008.

[21] Luca Benini and Giovanni de Micheli, "System-level power optimization: techniques and tools" *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, Volume 5(2), pp.115-192, April 2000.

[22] Jan M. Rabaey, Anantha P. Chandrakasan, Borivoje Nikolic, Digital integrated circuits: a design perspective, 2nd Ed. Prentice Hall,2008.

[23] Jan Rabaey, Low Power Design Essentials, Springer, 2009.

[24] K. Illgner et al., "Programmable DSP platform for digital still cameras," *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, Phoenix, AZ, USA, Vol.4, pp.2235-2238, March, 1999.

[25] W. Rabadi, R. Talluri, K. Illgner, J. Liang, and Y. Yoo, "Programmable DSP Platform for Digital Still Cameras," Application Report, Apr. 2000, TI.

[26] Hyun Lim, Soon-Young Park, Seong-Jun Kang and Wan-Hyun Cho, "FPGA Implementation of Image Watermarking Algorithm for a Digital Camera," *IEEE Pacific Rim Conference on Communications, Computers and signal Processing, 2003. PACRIM*. 2003, Vol.2, pp.1000-1003, Aug. 2003.

[27] Rongzheng Zhou et al., "System-on-Chip for Mega-Pixel Digital Camera Processor with Auto Control Fiinctions," *Proceedings of 5th International Conference on ASIC*, Vol.2, pp.894-897, Oct.2003.

[28] S. An C. Wang, "Recursive algorithm, architectures and FPGA implementation of the two-dimensional discrete cosine transform," *IET Image Processing*, vol. 2(6), pp. 286–294, 2008.

[29] Danian Gong, Yun He, and Zhigang Cao, "New Cost-Effective VLSI Implementation of a 2-D Discrete Cosine Transform and Its Inverse," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14(4), pp.405-415, April 2004.

[30] Shih-Chang Hsia and Szu-Hong Wang, "Shift-Register-Based Data Transposition for Cost-Effective Discrete Cosine Transform," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol.15(6), pp.725-728, June 2007.

[31] Che-Hong Chen, Bin-Da Liu, Jar-Ferr Yang, and Jiun-Lung Wang, "Efficient Recursive Structures for Forward and Inverse Discrete Cosine Transform," *IEEE Transactions on Signal Processing*, vol.52(9), pp.2665-2669, Sep. 2004.

[32] J. Eyre and J. Bier, "DSP processors hit the mainstream," *Computer*, vol.31 (8), pp.51-59, Aug 1998.

[33] A. Hoffmann et al., "A novel methodology for the design of application specific instruction set processors (ASIP) using a machine description language," *IEEE Transactions on Computer-Aided Design*, vol. 20, Nov. 2001, pp. 1338–1354.

[34] F. Sun, S. Ravi, A. Raghunathan, N. K. Jha. "A Scalable Synthesis Methodology for Application-Specific Processors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol.14 (11), Nov. 2006, pp 1175-1188.

[35] K. Keutzer, S. Malik, and A. R. Newton, "From ASIC to ASIP: The next design discontinuity," *Proceedings of International Conference on Computer Design*, Freiburg, Germany, pp. 84–90, Sep. 2002.

[36] O. Schliebusch, H. Meyr, and R. Leupers, Optimized ASIP Synthesis from Architecture Description Language Models, Springer, 2007.

[37] Y. Neuvo, "Cellular phones as embedded systems," *IEEE International Solid-State Circuits Conference, ISSCC*, Digest of Technical Papers, Vol.1, pp.32-37, Feb. 2004.

[38] Hao Xiao, An Pan, Yun Chen, and Xiaoyang Zeng, "Low-Cost Reconfigurable VLSI Architecture for Fast Fourier Transform," *IEEE Transactions on Consumer Electronics*, Vol. 54(4),pp.1617-1622, Nov. 2008.

[39] Hansoo Kim and In-Cheol Park, "High-Performance and Low-Power Memory-Interface Architecture for Video Processing Applications," *IEEE Transactions on Circuits And Systems for Video Technology*, Vol. 11(11), pp.1160-1170, Nov. 2001.

[40] Sven Wuytack, Francky Catthoor, Lode Nachtergaele and Hugo De Man, "Power Exploration for Data Dominated Video Application," *International Symposium on Low Power Electronics and Design*, 1996., pp. 359 – 364, Aug. 1996.

[41] E. Catthoor, E. Franssen, S. Wuytack, L .Nachtergaele and H.De Man "Global communication and memory optimizing transformations for low power signal processing systems," *Workshop on VLSI Signal Processing*, VII, 1994, pp.178-187, 1994.

[42] H.T. Kung, "Why systolic architectures?," *Computer* , Vol.15(1), Jan. 1982, pp.37-46.

[43] W. Ma, "2-D DCT systolic array implementation," *Electronics Letters* , Vol.27 (3), 31 Jan. 1991, pp.201-202.

[44] J. S. Ward and B. J. Stanier, "Fast discrete cosine transform algorithm for systolic arrays," *Electronics Letters* , Vol.19(2), Jan. 1983 pp.58-60.

[45] Nam Ik Cho and Sang Uk Lee, "DCT algorithms for VLSI parallel implementations," *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol.38 (1), Jan. 1990, pp.121-127.

[46] M.H. Lee and Y. Yasuda, "New 2D systolic array algorithm for DCT/DST," *Electronics Letters* , Vol.25 (25), 7 Dec. 1989, pp.1702-1704.

[47] M. Sun, L. Wu, M. Liou, "A concurrent architecture for VLSI implementation of discrete cosine transform," *IEEE Transactions on Circuits and Systems*, Vol.34 (8), Aug 1987, pp. 992- 994.

[48] M. H. Lee, "On computing 2-D systolic algorithm for discrete cosine transform," *IEEE Transactions on Circuits and Systems,* Vol. 37 (10), Oct 1990, pp.1321-1323.

[49] H. Malvar, "Fast computation of discrete cosine transform through fast Hartley transform," *Electronics Letters*, Vol. 22 (7), Mar. 1986, pp.352-353.

[50] Y.-M. Chin and J.-L. Wu, "Convolution-based DCT algorithm," *Electronics Letters*, Vol. 27 (20), Sept. 1991, pp.1834-1836.

[51] S. A. White, "Applications of distributed arithmetic to digital signal processing: a tutorial review," *IEEE ASSP Magazine*, vol.6, no.3, Jul.1989, pp.4-19.

[52] M.-T. Sun, T.-C. Chen, A.M. Gottlieb, ''VLSI Implementation of a 16x16 Discrete Cosine Transform," *IEEE Transactions on Circuits and Systems*, vol.36(4), pp. 610 – 617, Apr. 1989.

[53] Thucydides Xanthopoulos and Anantha P. Chandrakasan, "A Low-Power DCT Core Using Adaptive Bitwidth and Arithmetic Activity Exploiting Signal Correlations and Quantization," *IEEE Journal of Solid-State Circuits*, Vol. 35(5), pp.740-750, May 2000.

[54] Sungwook Yu and Earl E. Swartzlander Jr., "DCT implementation with distributed arithmetic," *IEEE Transactions on Computers*, vol.50(9), pp.985-991, Sep 2001.

[55] Chua-Chin Wang, Chia-Hao Hsu, Tuo-Yu Yao and Jian-Ming Huang, "A ROM-less DDFS Using A Nonlinear DAC With An Error Compensation Current Array," *IEEE Asia Pacific Conference on Circuits and Systems, APCCAS* 2008, pp.1632-1635, Nov. 30 2008-Dec. 3 2008.

[56] Chua-Chin Wang, Jian-Ming Huang, Y.-L. Tseng, Wun-Ji Lin and Ron Hu, "Phase-Adjustable Pipelining ROM-Less Direct Digital Frequency Synthesizer With a 41.66-MHz Output Frequency," *IEEE Transactions on Circuits and Systems*—II: Vol.53(10), Oct. 2006, pp.1143-1147.

[57] A. Shams, A. Chidanandan, W. Pan, and M. Bayoumi, "NEDA: A low power high throughput DCT architecture," *IEEE Transactions on Signal Processing*, Vol.54 (3), Mar. 2006, pp.955-964.

[58] Peng Chungan, Cao Xixin, Yu Dunshan, Zhang Xing, "A 250MHz optimized distributed architecture of 2D 8x8 DCT," *7th International Conference on ASIC*, pp. 189 – 192, Oct. 2007.

[59] Yuan-Ho Chen, Tsin-Yuan Chang and Chung-Yi Li, "High Throughput DA-Based DCT With High Accuracy Error-Compensated Adder Tree," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol.19(4), pp.709-714, April 2011.

[60] S.-F. Hsiao, Y.H. Hu, T.-B. Juang and C.-H. Lee, "Efficient VLSI Implementations of Fast Multiplierless Approximated DCT Using Parameterized Hardware Modules for Silicon Intellectual Property Design," *IEEE Transactions on Circuits and Systems—I*: Regular Papers, Vol. 52(8), pp.1568-1579, Aug. 2005.

[61] J. F. Yang and C. P. Fan, "Recursive implementation of discrete cosine transforms: with selectable fixed coefficient filters," *IEEE Transactions on Circuits and Systems. II*, Vol. 46, Feb. 1999, pp. 211–216.

[62] C.H. Chen, B.D. Liu, J.F.Yang and J.-L.Wang, "Efficient Recursive Structures for Forward and Inverse Discrete Cosine Transform," *IEEE Trans. on Signal Processing*, Vol. 52, No. 9, Sept. 2004, pp.2665-2669.

[63] J. F. Yang, and C. P Fan, "Compact recursive structures for discrete cosine transform," *IEEE Transactions on Circuits System II*, vol. 47,Apr.2000, pp. 314–321.

[64] C. Chen. B. Liu, and J. Yang, "Direct recursive structures for computing radix-r two-dimensional DCT/IDCT/DST/IDST," *IEEE Transactions on Circuits and Systems.-I*, Vol. 51(10), Oct. 2004, pp. 2017–2030.

[65] A. Elnaggar, and H.M. Alnuweiri, "A new multidimensional recursive architecture for computing the discrete cosine transform," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol.10 (1), Feb. 2000, pp.113-119.

[66] C.-T. Chiu and K.J.R. Liu, "Real-time parallel and fully pipelined two-dimensional DCT lattice structures with application to HDTV systems," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol.2(1), Mar 1992, pp.25-37.

[67] Yuk-Hee Chan, Lap-Pui Chau, and Wan-Chi Siu, "Efficient implementation of discrete cosine transform using recursive filter structure," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol.4(6), Dec. 1994, pp.550-552.

[68] Z. Qihui, C. Jianghua, Z. Shaohui and M. Nan, "A VLSI implementation of pipelined JPEG encoder for grayscale images," *Proc. International Symposium on Signals, Circuits and Systems*, pp.1-4, 2009.

[69] M. Kovac and N. Ranganathan, "JAGUAR: A fully pipelined VLSI architecture for JPEG image compression standard," *Proceedings of the IEEE*, Vol.83(2), pp. 247-258, 1995.

[70] M. Kovac, N. Ranganathan and M. Zagar, "A prototype VLSI chip architecture for JPEG image compression," *Proceedings of European Design and Test Conference*, 1995. ED&TC, pp.2-6., 1995.

[71] Sung-Hsien Sun and Shie-Jue Lee, "A JPEG Chip for Image Compression and Decompression," *Journal of VLSI Signal Processing,* Vol.35(1), pp.43–60, 2003.

[72] David Salomon and Giovanni Motta, Handbook of Data Compression. (5th Edition), Springer, 2010.

[73] Zhou Wang and Alan C. Bovik, Modern Image Quality Assessment, Morgan & Claypool Publishers, 2006.

[74] H. R. Sheikh, Alan C. Bovik and Gustavo de Veciana, "An Information Fidelity Criterion for Image Quality Assessment Using Natural Scene Statistics," *IEEE Transactions on Image Processing*, Vol. 14 (12), Dec. 2005, pp. 2117-2128.

[75] Zhou Wang and Alan C.Bovik, "Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures," *IEEE Signal Processing Magazine*, Vol. 26 (1), Jan. 2009, pp.98-117.

[76] T. Acharya and P. Tsai, JPEG2000 Standard for Image Compression. John Wiley and Sons, Inc., 2005.

[77] X. Kavousianos, E. Kalligeros, and D. Nikolos "Multilevel-Huffman Test-Data Compression for IP Cores With Multiple Scan Chains," *IEEE Transactions Very Large Scale Integration (VLSI) Systems*, Vol. 16 (7), July 2008, pp.926-931.

[78] P. G. Howard and J. S. Vitter, "Arithmetic Coding for Data Compression," *Proceedings of the IEEE*, Vol. 82 (6), June 1994.

[79] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE Transactions on Computers*, Vol. C-23, pp. 90-93, Jan. 1974.

[80] Cheng Chao and K.K Parhi, "Hardware Efficient Fast DCT Based on Novel Cyclic Convolution Structure," *IEEE Transactions on Signal Processing*, Vol. 54(11), Nov. 2006, pp.4419-4434.

[81] F . M. Bayer and R. J. Cintra, "Image Compression Via a Fast DCT Approximation," *IEEE Latin America Transactions*, Vol. 8 (6), Dec. 2010, pp.708-713.

[82] Il Dong Yun and Sang Uk Lee, "On the Fixed-Point-Error Analysis of Several Fast DCT Algorithms," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol.3 (1), Feb.1993, pp. 27-41.

[83] A. N. Skodras and A.G. Constantinides, "Efficient input-reordering algorithms for fast DCT," *Electronics Letters* , Vol.27 (21), Oct. 1991, pp.1973-1975.

[84] Jie Liang and T.D.Tran , "Fast multiplierless approximations of the DCT with the lifting scheme," *IEEE Transactions on Signal Processing*, Vol.49(12), Dec 2001, pp.3032-3044.

[85] H. S. Hou, "A fast recursive algorithm for computing the discrete cosine transform," *IEEE Transactions on Acoustics, Speech, Signal Processing*, Vol. ASSP-35 (10), Oct. 1987, pp. 1455–1461.

[86] S. C. Chan and K. L. Ho, "A new two-dimensional fast cosine transform algorithm," *IEEE Transactions on Signal Processing,* Vol.39 (2), Feb 1991, pp.481-485.

[87] Yu-Tai Chang and Chin-Liang Wang, "A New Fast DCT Algorithm and Its Systolic VLSI Implementation," *IEEE Transactions on Circuits And Systems—II: Analog and Digital Signal Processing*, Vol. 44 (11), Nov.1997, pp.959-962.

[88] David L. McLaren, D. Thong Nguyen, "Removal of subjective redundancy from DCT coded images," *IEE Proceedings I, Communications, Speech and Vision*, Vol.3, pp.482 – 485, 2001.

[89] P. K. Meher, T . Srikanthan, J. C. Patra, ''Scalable and Modular Memory-Based Systolic Architectures for Discrete Hartley Transform," *IEEE Transactions on Circuits and Systems*, Vol.53, no.5, pp. 1065 – 1077, May 2006.

[90] C. Moraga, "Generalized Discrete Hartley Transforms," 39th International Symposium on Multiple-Valued Logic, *ISMVL*, pp. 185 – 190, May 2009.

[91] Sabri A. Mahmoud, Ashraf S. Mahmoud, "The use of Hartley transform in OCR with application to printed Arabic character recognition," *Pattern Analysis & Applications*, Vol.12 (4), pp. 353-365, July.2008.

[92] R. N. Bracewell, O. Buneman, H. Hao, J. Villasenor, ''Fast Two-Dimensional Hartley Transform," *Proceedings of the IEEE*, Vol.74, pp. 1282 – 1283, Sept. 1986.

[93] S. K. Pattanaik, K. K. Mahapatra, "DHT Based JPEG Image Compression Using a Novel Energy Quantization Method," *IEEE International Conference on Industrial Technology*, pp.2827-2832, Dec.2006.

[94] A. B. Watson , and A. Poirson, ''Separable two-dimensional discrete Hartley transform," *Journal of the Optical Society of America A*, Vol.3(12), Dec. 1986.

[95] A. Amira, "An FPGA based System for Discrete Hartley transforms," *International Conference on Visual Information Engineering*, pp. 137 – 140, 7-9 July 2003.

[96] Web, www.xilinx.com

[97] Xilinx Inc., User Guide, ChipScope Pro Software and Cores.

[98] Jian, Bian Li, Xuan, Zeng, Rong, Tong Jia and Yue, Liu: 'An Efficient VLSI Architecture For 2D-DCT Using Direct Method', *Proceedings. 4th International Conference on ASIC*, 2001, Fudan Univ., Shanghai, 2001, pp.393-396.

Academic

- B.Tech (2004-2008) in Electronics and Communication Engineering from SASTRA Deemed University, Thanjavur, Tamilnadu, India.
- Admitted for M.Tech (Research) degree in July 2009 in Electronics and Communication Engineering, NIT Rourkela, India.

Work Experience

- Worked as Engineer Trainee at Honeywell Technology Solutions Lab, Madurai from 29 Sept. 2009 to 30 Jan. 2010.
- Worked in SMDP-II VLSI Project at NIT Rourkela from 1 Aug. 2011 to 17 July 2012.

Publications Related to the Thesis

In Journals

[1] **Vijay Kumar Sharma**, Umesh C. Pati and K. K. Mahapatra, "A Simple VLSI Architecture for Computation of 2-D DCT, Quantization and Zig-zag ordering for JPEG," *International Journal of Signal and Imaging Systems Engineering (IJSISE)*, Vol 5 (1), pp.58-65, 2012.

[2] **Vijay Kumar Sharma**, K. K. Mahapatra and Umesh C. Pati, "Non-Recursive Equation and Direct Computation of 8x8 2-D DCT Coefficients for High Accuracy and Low Hardware," *Integration, the VLSI Journal* (**Revised version to be submitted**) .

In International Conferences

[1] **Vijay Kumar Sharma**, Richa Agrawal, U. C.Pati, and K. K. Mahapatra, "2-D Separable Discrete Hartley Transform Architecture for Efficient FPGA Resource," *International Conference on Computer and Communication Technology, ICCCT* 2010, MNNIT Allahabad, pp.236-241,17-19 Sept. 2010.

[2] **Vijay Kumar Sharma**, U. C. Pati and K. K. Mahapatra, "An Study of Removal of Subjective Redundancy in JPEG for Low Cost, Low Power, Computation efficient Circuit Design and High Compression Image," *International Conference on Power, Control and Embedded Systems, ICPCES* 2010, MNNIT Allahabad, pp. 1-6, Nov. 29 -Dec. 1, 2010.

[3] **Vijay Kumar Sharma**, U. C. Pati and K. K. Mahapatra, "A Simple VLSI Architecture for Computation of 2-D DCT, Quantization and Zig-zag ordering for JPEG," *International Conference on Electronic Systems, ICES* 2010, NIT Rourkela, Jan. 7-9, pp. 182-185.

[4] **Vijay Kumar Sharma**, K. K. Mahapatra and Umesh C. Pati, "An Efficient Distributed Arithmetic based VLSI Architecture for DCT," *International Conference on Devices and Communications*, Feb. 24-25, 2011, BIT Mesra, Ranchi, pp. 1-5.