AN ABSTRACT OF THE THESIS OF

Dwight Poplin for the degree of Master of Science in Electrical and Computer Engineering presented on May 2, 1997. Title: Distributed Arithmetic Architecture for the Discrete Cosine Transform.

Abstract approved: Redacted for Privacy

Sayfe Kiaei

The Discrete Cosine Transform is used in many image and video compression standards. Many methods have been developed for efficiently computing the Discrete Cosine Transform including flowgraph algorithms, distributed arithmetic and two-dimensional decompositions.

A new architecture based on distributed arithmetic is presented for computing the Discrete Cosine Transform and it's inverse. The main objective of the design is to minimize the area of the VLSI implementation while maintaining the throughput necessary for video and image compression standards such as MPEG and JPEG. Several improvements have been made compared to previously published distributed arithmetic architectures. These include elimination of four lookup tables and implementation of the lookup tables using logic instead of ROM.

A model of the proposed architecture was written in C. The model was used to verify the accuracy of the architecture and to do JPEG compression on a series of test images. Behavioral simulations were performed with a hardware model written in the Verilog hardware description language. These behavioral simulations verify that the hardware implementation matches the C model. The model was synthesized using the Synopsis synthesis tool. The gate count and clock rate of the design were estimated using the synthesis results.

Distributed Arithmetic Architecture for the Discrete Cosine Transform

by

Dwight Poplin

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented May 2, 1997
Commencement June 1997

Master of Science thesis of Dwight Poplin presented on May 2, 1997

APPROVED:

Redacted for Privacy

Major Professor, representing Electrical and Computer Engineering

Redacted for Privacy

Head of Department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for Privacy

Dwight Poplin, Author

ACKNOWLEDGEMENT

I must begin by thanking Prof. Sayfe Kiaei, my advisor and mentor. When I began working in the electronics industry, I always hoped I would some day get the chance to probe the mysteries of DSP. Prof. Kiaei's classes have been great and his help with this research is deeply appreciated. I would like to thank Prof. Ben Lee and Prof. David Allstot who served on my committee. The quality of instruction I have received from all of these professors has been excellent and I appreciate the effort they put into their teaching.

I am grateful to several of my coworkers for their help. Dr. Jon Gibson met with me regularly to keep this research focused. He contributed many valuable ideas, including the idea to implement the lookup tables using logic. Matt Heineck spent a lot of time teaching me how to use a new set of VLSI tools. I'm fortunate that he was so willing and able to help. I also appreciate the contributions of Daryl Anderson, who helped arrange funding for my research and Trung Vo-Vu, whose JPEG C models are amazingly easy to use.

Thanks to Yu Zhi Chao for introducing me to the Discrete Cosine Transform and Distributed Arithmetic. Thanks to Amit Dutta, Navid Lashkadarian, Sumit Talwalker and Takao Inoue who have all taken time to share their knowledge.

On a personal note, I'm indebted to Sue Kuenzi for her encouragement and emotional support during the last two years. Graduate school would have been a lot more difficult without her. Last, but not least, I'm grateful to God for giving me the opportunity and ability to pursue higher education. It has been a privilege.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

DISTRIBUTED ARITHMETIC ARCHITECTURE FOR THE DISCRETE COSINE TRANSFORM

IMAGE AND VIDEO COMPRESSION OVERVIEW

During the last decade, the demand for storage, transmission, and real-time processing of images and video has grown rapidly. This growth has been fueled by the development of multimedia software for the personal computer, exchange of images on the Internet, the development of high-definition television (HDTV) and the availability of high-capacity storage devices such as the compact disk and the emerging digital video disk (DVD).

Representing images and video in digital form requires a great deal of data. Consider for example a single color VGA image for a personal computer. The image is 640 pixels wide by 480 pixels tall. Since it has 3 color components, it requires $640 \times 480 \times 3 = 921,600$ bytes. The requirements are even greater for video. If color VGA images are sent at a rate of 30 frames per second, one second of video would require 27.6 Megabytes and a 90 minute movie would require 149 Gigabytes! Because of the sheer size of images and video, several compression standards have been developed in the last decade.

The Joint Photographic Experts Group has a defined an image compression standard known as JPEG. A block diagram of a JPEG coder is shown in Figure 1.1. The Motion Picture Experts Group has developed a family of standards, known as MPEG, for video compression. A block diagram of an MPEG coder is shown in Figure 1.2. The International Telecommunications Union (ITU) has defined the H.263 standard for very-low bit rate video applications such as video telephones. These standards are based on transform coding using the Discrete Cosine Transform (DCT) and the Inverse Discrete Cosine Transform (IDCT).

The JPEG standard achieves compression by exploiting the intraframe redundancy inherent in images. Intraframe redundancy means that the magnitude of a picture element, or pixel, tends to be the same or close to the magnitude of pixels which are vertically and horizontally adjacent to it in the same frame. The DCT plays a critical role in exploiting intraframe redundancy. In addition to using *intra*frame redundancy, MPEG and H.263 achieve further compression by exploiting the *inter*frame redundancy of video. Interframe redundancy means that the magnitude of a pixel tends to be the same or close to the magnitude of a pixel in the same position in the previous or following frame.

The JPEG Still Image Compression Standard

The key elements of the JPEG coder in Figure 1.1 are the DCT, Quantizer, Run-Length Coder and Huffman Coder. JPEG operates on one 8x8 block of pixels at a time. The first step in JPEG image compression is to divide the image into blocks of this size. If the horizontal or vertical dimension of the image is not evenly divisible by 8, the last rows or columns of the image are duplicated to pad the blocks out to an even boundary. If a color image is being compressed, each of the three color planes is compressed separately.

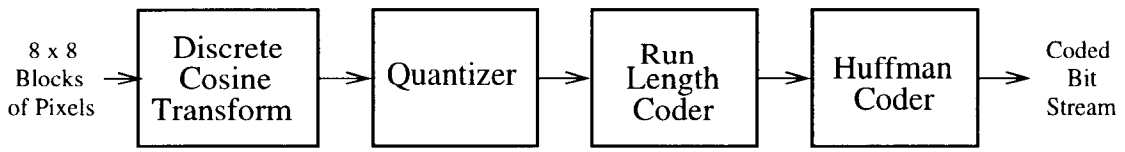


Figure 1.1: JPEG Coder Block Diagram

The DCT transforms each block from the spatial domain into an 8x8 block of frequency domain coefficients. The low-frequecy components contain significant energy, but the other coefficients tend to be close to zero. Based on this observation, the DCT coefficients are selectively quantized, with the high frequency coefficients being quantized more coarsely than the DC and low frequency coefficients. An 8x8

matrix of quantization constants must be supplied to the coder for each image. This matrix is the most important factor in determining compression ratio and coded image quality. Quantization is equivalent to doing an integer division of the DCT coefficients by the quantization matrix.

After quantization, the 8x8 block tends to have many zeros in the medium and high-frequency areas. The coefficients are converted to a sequence of 64 words by scanning them out in zig-zag fashion starting at the upper left of the matrix and finishing at the lower right. This scanning order means that long strings of zeros will be encountered toward the end of each block. The zeros are run-length coded to compress the sequence. The run length coded word stream is then Huffman coded to achieve further compression. A more complete explanation of the JPEG standard can be found in [33] and in [26].

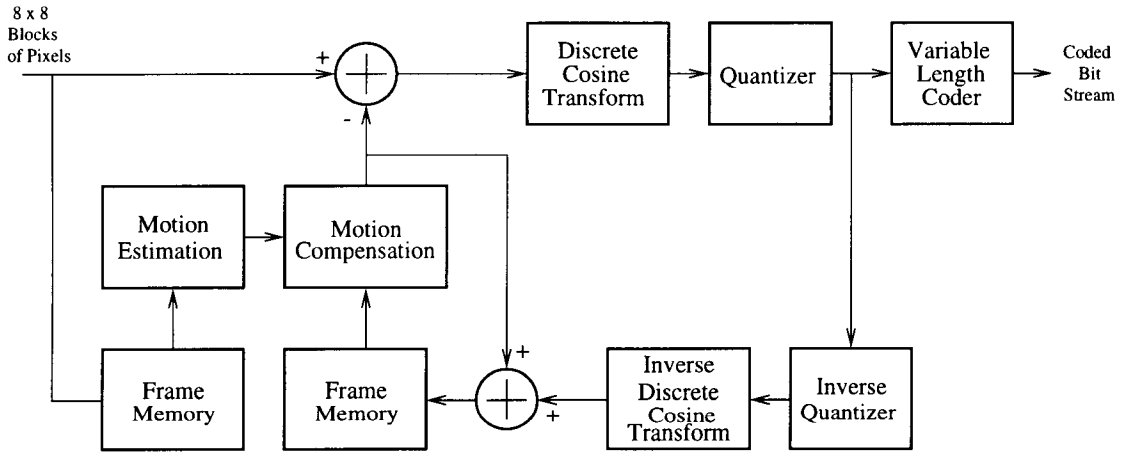


Figure 1.2: MPEG Video Coder Block Diagram

The MPEG Video Compression Standard

Figure 1.2 shows a block diagram of an MPEG video coder. Intraframe coding is done by the three blocks in the upper right part of the diagram, which are essentially a JPEG coder. Interframe coding is done using a feedback loop similar to Differential Pulse Code Modulation. For each pixel, the value of that pixel in the previous

frame is subtracted from the current value, so that only the difference is coded. An estimate of the decoded signal is generated within the coder by the inverse quantizer and IDCT. The coder also estimates the motion of objects within the current frame and generates displacement vectors which compensate for this motion. A further explanation of the MPEG standard can be found in [25] and in [23].

The Discrete Cosine Transform

The 2-D DCT transforms a block of N x N pixels from the spatial domain into the frequency domain. (It is possible to perform a DCT on a matrix which is not square, but this is rarely if ever done in image and video coding.) By examining the image in the frequency domain, the spectral components can be identified. Because of intraframe redundancy, the higher frequency components tend to have a low magnitude and do not contribute significantly to the resolution of the image.

The definition of an N x N 2-D DCT is:

$$Y(k,l) = \frac{2}{N}\,\alpha(k)\,\alpha(l)\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} X(m,n)\;\; cos\left[\frac{(2m+1)k\pi}{2N}\right]\; cos\left[\frac{(2n+1)l\pi}{2N}\right] \quad (1.1)$$

where X(m,n) is an N x N input matrix taken from the image, Y(k,l) is an N x N matrix of frequency domain coefficients, and $\alpha(n)$ is a vector of scaling constants which ensure the orthogonality of the transform given by

$$\alpha(n) = \begin{pmatrix} \frac{1}{\sqrt{2}}, & n = 0 \\ 1, & n \neq 0 \end{pmatrix}$$

The N x N 2-D IDCT is given by:

$$X(m,n) = \frac{2}{N}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1}\alpha(k)\,\alpha(l)\,Y(k,l)\;\; cos\left[\frac{(2m+1)k\pi}{2N}\right]\; cos\left[\frac{(2n+1)l\pi}{2N}\right] \quad (1.2)$$

The DCT finds widespread use because it has a high energy compaction property and can be computed without prior knowledge of the statistics of the input signal. The inputs and outputs are also real numbers, rather than complex numbers as

in the Discrete Fourier Transform. The relative advantages of the DCT in image processing applications are described in Section 7.8 of Rao and Yip [26].

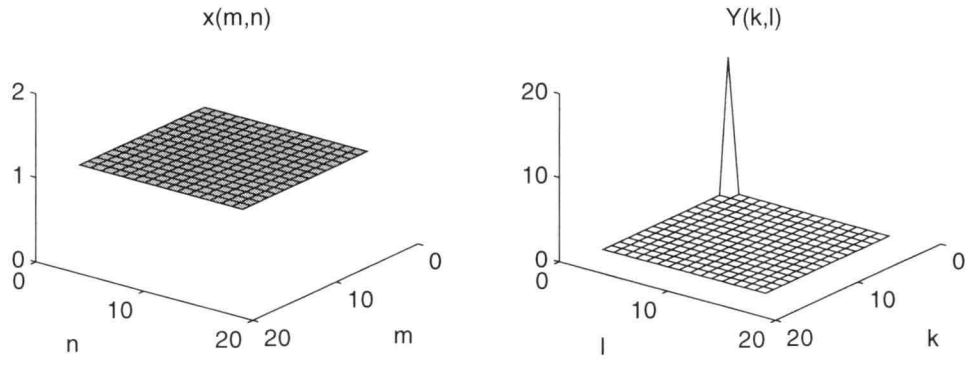


Figure 1.3: DCT Coefficients for DC Input

It is helpful to have an intuitive understanding of the 2-D DCT. The 2-D DCT transforms the input matrix into a matrix of frequency domain coefficients. The meaning of these coefficients can be appreciated by considering the series of signals shown in Figures 1.3 through 1.8. Each figure shows an input block, x(m,n), and the corresponding DCT coefficients, Y(k,l). First consider a DC input, shown in Figure 1.3, given by

$$X(m,n) = 1 \quad \forall\ m,n$$

The coeffients are

$$Y(k,l) = \begin{pmatrix} N, & k=l=0 \\ 0, & else \end{pmatrix}$$

This figure shows that the Y(0,0) coefficient is the average DC value of the input multiplied by a factor of N. With reference to a block of image data, this position in the matrix will be referred to as the upper left hand corner.

Figure 1.4 shows the DCT output for three signals which vary only in the horizontal dimension. For these graphs,

$$X(m,n) = cos\left(\frac{2\pi f(n+0.5)}{N}\right), \qquad \begin{aligned} n,m &= 0\ ..\ N-1 \\ f &= 1,2,4 \end{aligned}$$

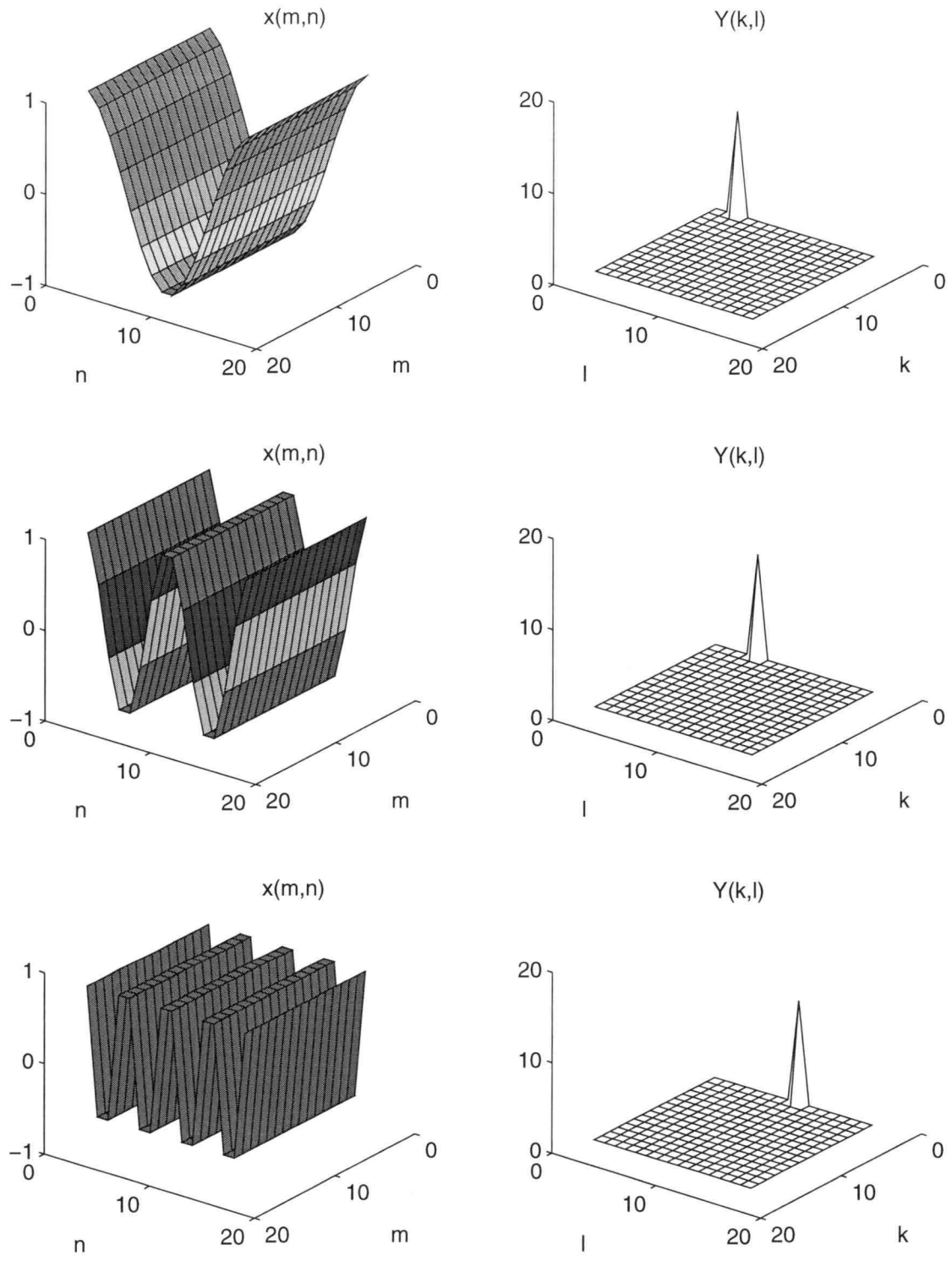


Figure 1.4: DCT Coefficients for Horizontally Varying Inputs

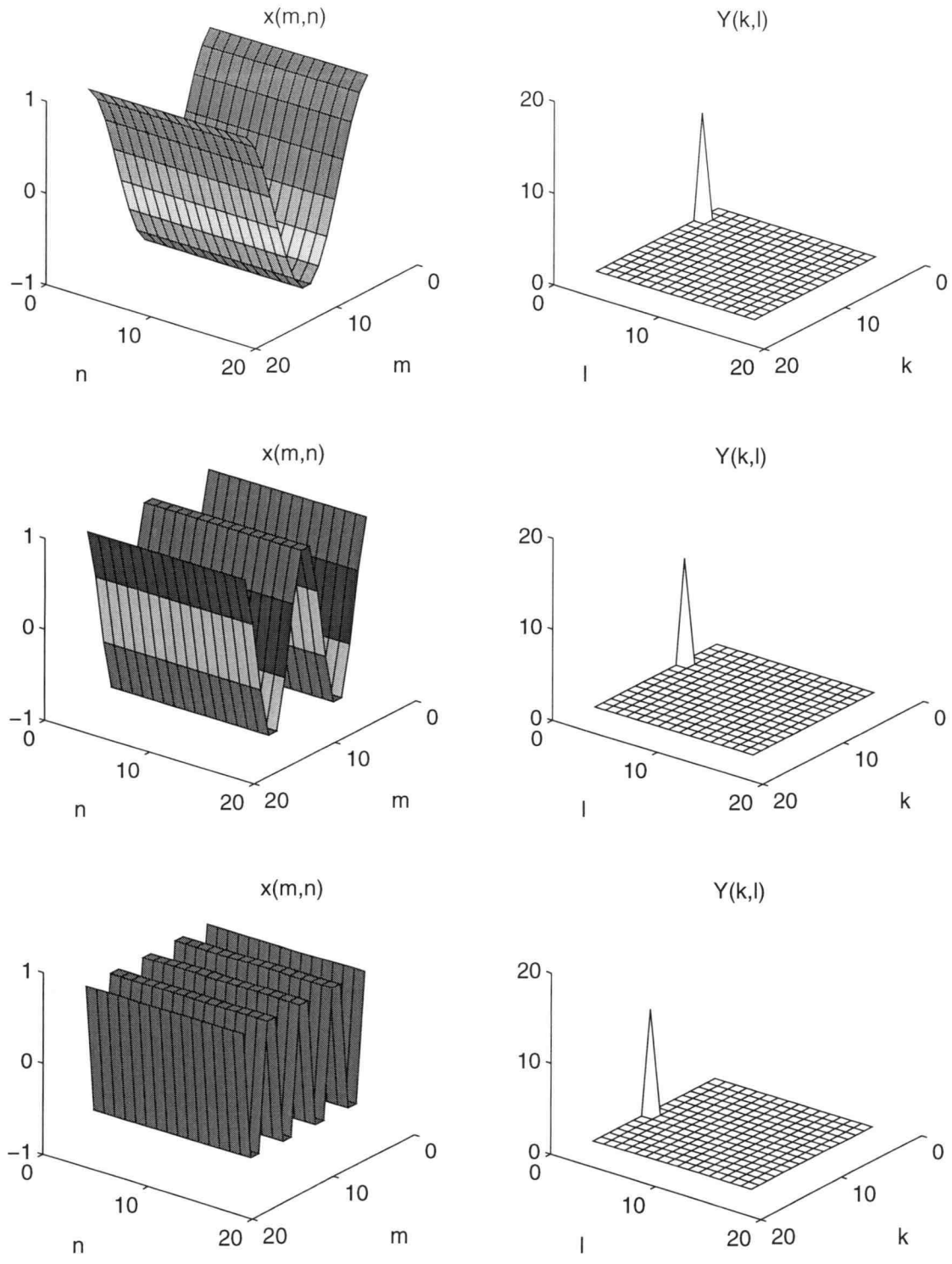


Figure 1.5: DCT Coefficients for Vertically Varying Inputs

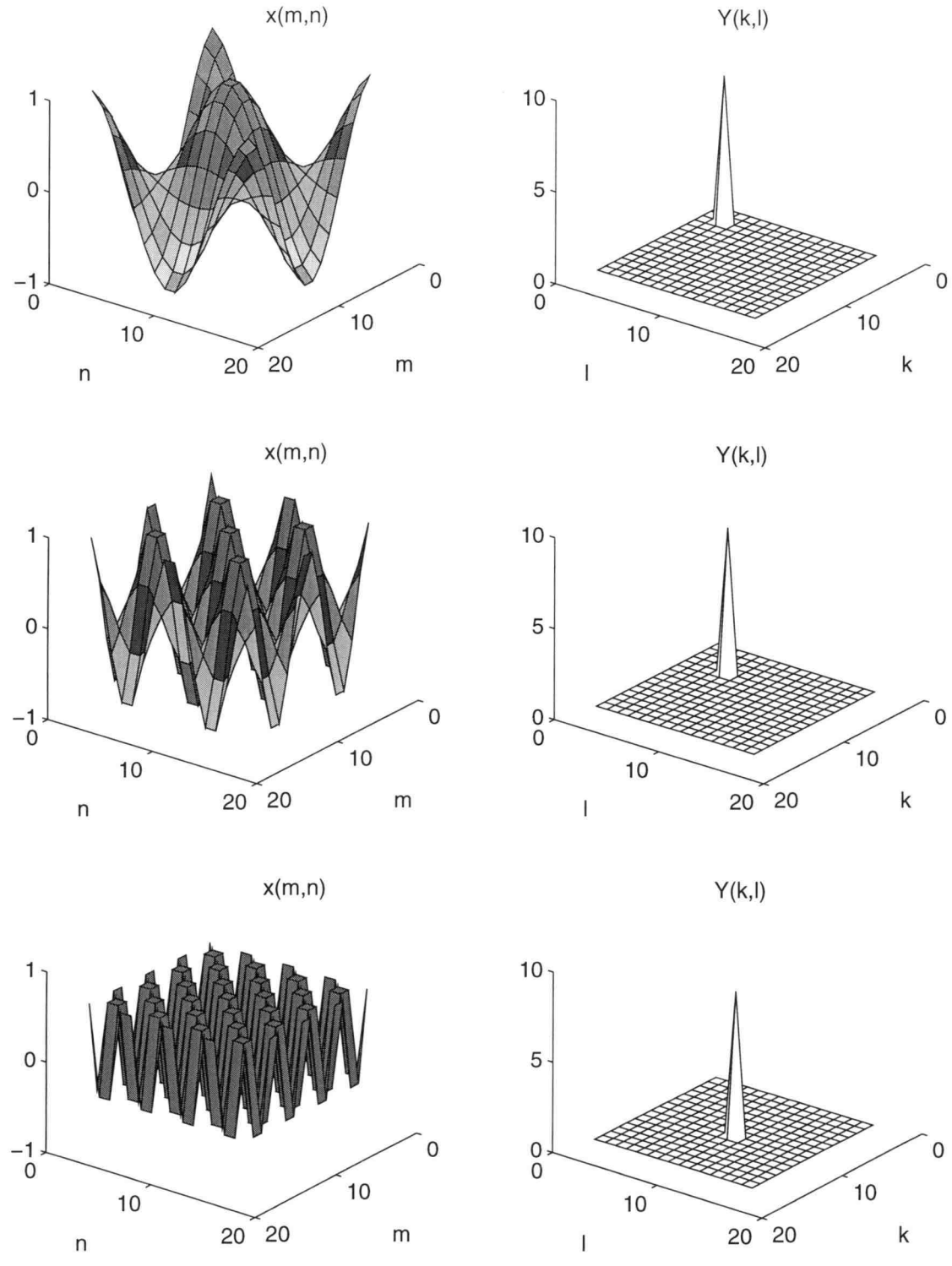


Figure 1.6: DCT Coefficients for Inputs Which Vary Horizontally and Vertically

and

$$Y(k,l) = \begin{pmatrix} \frac{N}{\sqrt{2}} & k = 0,\ l = 2f \\ 0 & else \end{pmatrix}$$

These graphs show that the horizontal frequency components of the signal appear along the top of the coefficient matrix. The coefficients toward the left of the matrix represent lower horizontal frequency and the coefficients toward the right represent higher horizontal frequency. The term "frequency" is used somewhat loosely here: rather than cycles per unit time, it refers to cycles per unit distance.

Figure 1.5 shows the DCT output for three signals which vary only in the vertical dimension. For these graphs,

$$x(m,n) = cos\left(\frac{2\pi f(m+0.5)}{N}\right), \qquad \begin{aligned} n, m &= 0\ ..\ N-1 \\ f &= 1, 2, 4 \end{aligned}$$

and

$$Y(k,l) = \begin{pmatrix} \frac{N}{\sqrt{2}} & k = 2f,\ l = 0 \\ 0 & else \end{pmatrix}$$

These graphs show that vertical frequency components of the signal appear along the left of the coefficient matrix. The coefficients toward the top of the matrix represent lower horizontal frequency and the coefficients toward the bottom represent higher horizontal frequency.

Figure 1.6 shows the DCT output for three signals which vary in both the vertical and horizontal dimensions. For these graphs,

$$x(m,n) = cos\left(\frac{2\pi f(m+0.5)}{N}\right) cos\left(\frac{2\pi f(n+0.5)}{N}\right), \qquad \begin{aligned} n, m &= 0\ ..\ N-1 \\ f &= 1, 2, 4 \end{aligned}$$

and

$$Y(k,l) = \begin{pmatrix} \frac{N}{2} & k = l = 2f \\ 0 & else \end{pmatrix}$$

These graphs show that the DCT coefficients for signal components which come from both horizontal and vertical frequencies show up in the lower right part of the matrix.

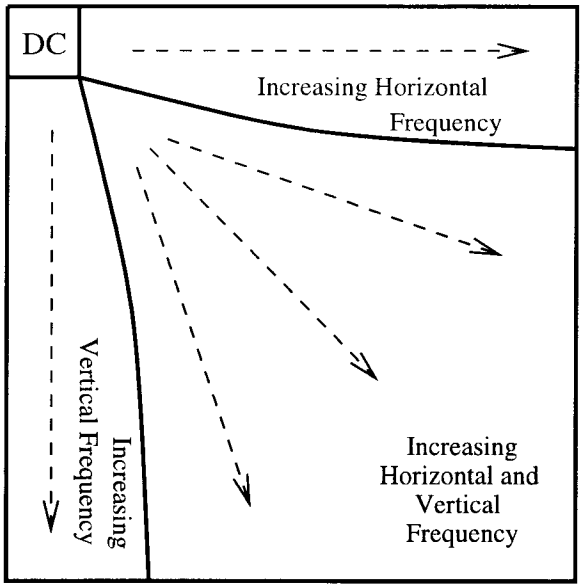


Figure 1.7: Partitioning of the DCT Coefficient Matrix

To summarize, Figure 1.7 shows that the coefficient matrix can be partitioned into areas of DC, horizontal frequencies, vertical frequencies and combinations of horizontal and vertical. The DC component of the input signal appears in the upper left corner of the coefficient matrix. Moving from left to right along the top row of the matrix is increasing horizontal frequency. Moving from top to bottom in the left column of the matrix is increasing vertical frequency. Moving from the top left of the matrix toward the lower right is increasing horizontal and vertical frequency.

When viewed in the frequency domain, intraframe redundancy means that the coefficients of the high-frequency components tend to have lower magnitude than the low-frequency components. This is illustrated in Figure 1.8 which shows a block from an image and its DCT coefficients.

Design Goals

Referring to equations 1.1 and 1.2, it can be seen that implementing an NxN 2-D DCT or IDCT directly from the definition requires N^2 multiplications and

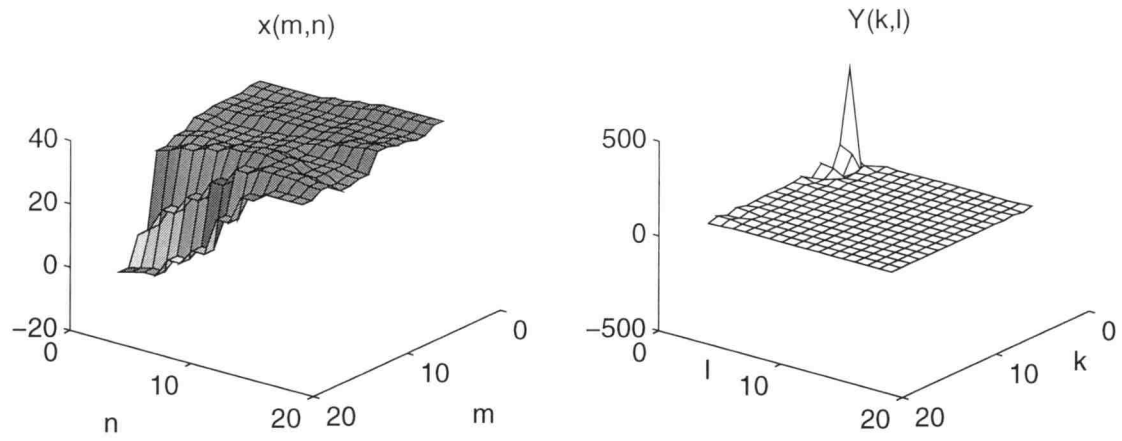


Figure 1.8: DCT Coefficients for Typical Image Input

additions per input sample. An input sample is defined as one pixel for the DCT or one coefficient for the IDCT. Since there are N^2 input samples, a total of N^4 multiplications and additions are required for one 2-D transform. Because of this complexity, the DCT is usually the bottleneck in a JPEG codec and one of the major bottlenecks in an MPEG codec.

The major goal of this research was to reduce the area of the DCT and IDCT relative to an existing JPEG codec while maintaining accuracy and throughput comparable to the existing design. The DCT/IDCT block in the existing design uses 11,200 gates and has a throughput of 7.76 Megasamples/second at a clock rate of 33 MHz. This throughput can also be expressed as 4.25 clock cycles per sample. To achieve the same throughput with direct implementation would require N^2 multiplications every 4.25 clock cycles, or about 15 multiplications per clock cycle. In a VLSI implementation, 15 single-cycle multipliers would require much more than 11,200 gates. To reduce the area of the DCT and IDCT, a less complex algorithm for computing these transforms must be used.

DISCRETE COSINE TRANSFORM IMPLEMENTATION METHODS

The Discrete Cosine Transform plays a central role in most video and image compression applications. In the previous chapter it was shown that the complexity of implementing an N x N 2-D DCT or IDCT directly is too great for a practical VLSI circuit. To reduce the area of these transforms, a less complex algorithm must be used. Because of the growing importance of image and video compression, a great deal of research has been done in this area and a number of algorithms have been discovered. These algorithms will be discussed in this chapter.

The chapter begins with an explanation of the row/column decomposition, a technique which calculates the 2-D DCT using the 1-D DCT. After defining the 1-D DCT, this chapter presents several techniques for reducing the complexity of the 1-D DCT. These include flowgraph algorithms, distributed arithmetic and Sheu's lookup table algorithm. These algorithms are compared based on their suitability for achieving minimum area.

Row-Column Decomposition

One of the properties of the 2-D DCT and IDCT is that they are separable transforms. This means that the 2-D DCT can be implemented using the 1-D DCT and the 2-D IDCT can be implemented using the 1-D IDCT. Because of separability, any algorithm which reduces the complexity of the 1-D DCT will also reduce the complexity of the 2-D DCT. In order to understand separability, it is necessary to first define the 1-D DCT and IDCT. A 1-D DCT is given by

$$y(k) = \sqrt{\frac{2}{N}}\ \alpha(k)\ \sum_{n=0}^{N-1}\ x(n)\ \ cos\left[\frac{(2n+1)k\pi}{2N}\right] \tag{2.1}$$

and a 1-D IDCT is given by

$$x(n) = \sqrt{\frac{2}{N}}\ \sum_{n=0}^{N-1}\ \alpha(k)\ y(k)\ \ cos\left[\frac{(2n+1)k\pi}{2N}\right] \tag{2.2}$$

where x is an Nx1 vector of input pixels and y is an Nx1 vector of 1-D DCT coefficients.

Separability is a way of calculating a 2-D transform using a 1-D transform. Specifically, separability means that the 2-D transforms can be performed by doing a 1-D transform on each row of the input matrix and then doing a 1-D transform on each column of the intermediate result. This technique is often called a row/column decomposition. For example, consider an NxN DCT input matrix X, where X_i is the i^{th} row vector of X. The first step in the row/column decomposition is to form an intermediate matrix U where U_i, the i^{th} row vector of U, is given by

$$U_i(k) = \sqrt{\frac{2}{N}}\ \alpha(k)\ \sum_{n=0}^{N-1} X_i(n)\ \ cos\left[\frac{(2n+1)k\pi}{2N}\right] \tag{2.3}$$

Next, the NxN 2-D DCT coefficient result, Y, is obtained by

$$Y_j(k) = \sqrt{\frac{2}{N}}\ \alpha(k)\ \sum_{n=0}^{N-1} U_j(n)\ \ cos\left[\frac{(2n+1)k\pi}{2N}\right] \tag{2.4}$$

where Y_j is the j^{th} column vector of Y and U_j is the j^{th} column vector of U.

Implementing an N x N 2-D DCT or IDCT using the row/column decomposition requires $2N$ 1-D transforms: N 1-D transforms for the rows and N for the columns. Referring to equations 2.1 and 2.2, each 1-D transform requires N^2 multiplications and additions. This is true because there are N points in the transform and each point requires N multiplications and additions. Therefore, an N x N 2-D DCT or IDCT using the row/column decomposition and direct implementation of the 1-D transform requires a total of $2N(N^2) = 2N^3$ multiplications and additions. This is a factor of $\frac{N}{2}$ reduction in complexity compared to direct implementation of the 2-D transform.

For some applications it is convenient to express a 1-D DCT in matrix form. Consider for example an 8-point 1-D DCT. We begin by defining

$$C_k = \sqrt{\frac{2}{N}}\ cos\left(\frac{k\ \pi}{N}\right) = 0.5\ cos\left(\frac{k\ \pi}{16}\right) \tag{2.5}$$

Using this notation, the second point in the DCT, y(1), can be written as

$$y(1) = \begin{bmatrix} C_1 & C_3 & C_5 & C_7 & C_9 & C_{11} & C_{13} & C_{15} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix}$$

To calculate all the points of the 1-D DCT, a matrix of cosine coefficients, Γ, is defined so that the 1-D DCT can be written as

$$y = \Gamma\, x$$

where

$$\Gamma = \begin{bmatrix} C_4 & C_4 & C_4 & C_4 & C_4 & C_4 & C_4 & C_4 \\ C_1 & C_3 & C_5 & C_7 & C_9 & C_{11} & C_{13} & C_{15} \\ C_2 & C_6 & C_{10} & C_{14} & C_{18} & C_{22} & C_{26} & C_{30} \\ C_3 & C_9 & C_{15} & C_{21} & C_{27} & C_{33} & C_{39} & C_{45} \\ C_4 & C_{12} & C_{20} & C_{28} & C_{36} & C_{44} & C_{52} & C_{60} \\ C_5 & C_{15} & C_{25} & C_{35} & C_{45} & C_{55} & C_{65} & C_{75} \\ C_6 & C_{18} & C_{30} & C_{42} & C_{54} & C_{66} & C_{78} & C_{90} \\ C_7 & C_{21} & C_{35} & C_{49} & C_{63} & C_{77} & C_{91} & C_{105} \end{bmatrix}$$

and $\Gamma_{k,n}$, the element in the k^{th} row and n^{th} column of Γ, is given by

$$\Gamma_{k,n} = \begin{cases} \sqrt{\frac{1}{N}}, & k = 0,\ 0 \le n \le N-1 \\ C_{(2n+1)k}, & 1 \le k \le N-1,\ 0 \le n \le N-1 \end{cases} \tag{2.6}$$

Taking advantage of the identities

$$cos(x + 2\pi) = cos(x)$$

$$cos(x) = cos(-x)$$

and

$$cos(x) = -cos(\pi - x)$$

Γ can be simplified so that an 8-point 1-D DCT can be written in matrix form as

$$\begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \\ y(4) \\ y(5) \\ y(6) \\ y(7) \end{bmatrix} = \begin{bmatrix} C_4 & C_4 & C_4 & C_4 & C_4 & C_4 & C_4 & C_4 \\ C_1 & C_3 & C_5 & C_7 & -C_7 & -C_5 & -C_3 & -C_1 \\ C_2 & C_6 & -C_6 & -C_2 & -C_2 & -C_6 & C_6 & C_2 \\ C_3 & -C_7 & -C_1 & -C_5 & C_5 & C_1 & C_7 & -C_3 \\ C_4 & -C_4 & -C_4 & C_4 & C_4 & -C_4 & -C_4 & C_4 \\ C_5 & -C_1 & C_7 & C_3 & -C_3 & -C_7 & C_1 & -C_5 \\ C_6 & -C_2 & C_2 & -C_6 & -C_6 & C_2 & -C_2 & C_6 \\ C_7 & -C_5 & C_3 & -C_1 & C_1 & -C_3 & C_5 & -C_7 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix} \tag{2.7}$$

and an 8-point 1-D IDCT can be written as

$$x = \Gamma^T y$$

or

$$\begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix} = \begin{bmatrix} C_4 & C_1 & C_2 & C_3 & C_4 & C_5 & C_6 & C_7 \\ C_4 & C_3 & C_6 & -C_7 & -C_4 & -C_1 & -C_2 & -C_5 \\ C_4 & C_5 & -C_6 & -C_1 & -C_4 & C_7 & C_2 & C_3 \\ C_4 & C_7 & -C_2 & -C_5 & C_4 & C_3 & -C_6 & -C_1 \\ C_4 & -C_7 & -C_2 & C_5 & C_4 & -C_3 & -C_6 & C_1 \\ C_4 & -C_5 & -C_6 & C_1 & -C_4 & -C_7 & C_2 & -C_3 \\ C_4 & -C_3 & C_6 & C_7 & -C_4 & C_1 & -C_2 & C_5 \\ C_4 & -C_1 & C_2 & -C_3 & C_4 & -C_5 & C_6 & -C_7 \end{bmatrix} \begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \\ y(4) \\ y(5) \\ y(6) \\ y(7) \end{bmatrix} \tag{2.8}$$

From this representation, it is evident that the IDCT is calculated using a matrix which is the transpose of the matrix used for the DCT. This is a characteristic of orthogonal transforms.

A 2-D DCT performed by the row/column decomposition can be written in matrix form as:

$$Y = \Gamma \times (X \times \Gamma^T) \tag{2.9}$$

where X is an N x N matrix of input pixels and Y is an N x N matrix of DCT coefficients. In similar fashion, a 2-D IDCT can be written as

$$X = \Gamma^T \times (Y \times \Gamma) \tag{2.10}$$

Because the 1-D DCT and IDCT are used to compute the 2-D DCT and IDCT, a great deal of research has been done to find efficient ways to implement them in VLSI. Most of these techniques can be classified either as flowgraph algorithms or as distributed arithmetic algorithms.

Flowgraph Algorithms

Flowgraph decomposition is the same technique which was used to derive the Fast Fourier Transform from the Discrete Fourier Transform. The basic idea is to decompose an N-point transform into 2 transforms of length $\frac{N}{2}$. Since an N-point DCT requires N^2 multiplications and additions, two $\frac{N}{2}$ -point DCT's require $2\left(\frac{N}{2}\right)^2$ or $\frac{N^2}{2}$ multiplications and additions. If this technique is applied recursively, the original N-point DCT can be reduced to $\frac{N}{2}$ 2-point DCT's. The number of multiplications and additions required to compute the transform falls from N^2 to roughly $Nlog_2N$.

Numerous DCT flowgraph algorithms have been reported. Chen's algorithm [3] was the first widely published flowgraph architecture. It required 16 multiplies and 26 additions for an 8 point 1-D DCT. Lee's algorithm [14] requires 12 multiplications and 29 additions. Since multipliers are more complex than adders, this was a substantial reduction in complexity. Jain [10] demonstrated a flowgraph which requires 16 multiplications and 32 additions but has lower dynamic range requirements in the internal nodes and greater accuracy. Hou's algorithm [12] uses 12 multiplications and 29 additions and is more regular than Lee's. Loeffler [18] reduced the number of multiplications to 11, with 29 additions. Linzer [17] further reduced the complexity to 10 multiplications and 26 additions by allowing the DCT coefficient outputs to be scaled by scaling factors. Cismas [5] also presents a scaled algorithm with 10 multiplies and 26 additions, but the scaling factors are all powers of two, making it easy to recover the exact coefficient value by a shift operation. A cogent summary of flowgraph algorithms can be found in Loeffler [18].

Table 2.1: A Comparision of Flowgraph Algorithms for the 8-Point 1-D DCT

Algorithm	Chen	Lee	Hou	Loeffler	Linzer	Cismas
Multiplications (1-D)	16	12	12	11	10	10
Additions (1-D)	26	29	29	29	26	26
Multiplications (2-D)	256	192	192	176	160	160
Additions (2-D)	416	464	464	464	416	416

Table 2.1 summarizes the number of multiplications and additions for these flowgraph algorithms. The top two rows show the number of multiplications and additions required for a single 1-D transform. The bottom two rows show the total number of multiplications and additions required for a 2-D transform, assuming that the 1-D algorithm is used together with the row/column decomposition. In addition to the number of multiplications and additions, there are several other important criterion for evaluating flowgraphs such as the dynamic range required for all internal nodes, the number of additions which can be performed as multiply-accumulates and the number of unique constants required.

As a example of a flowgraph algorithm, consider the algorithm described by Chen [3]. Using the first level of decomposition from this algorithm, an 8-point 1-D DCT

$$\begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \\ y(4) \\ y(5) \\ y(6) \\ y(7) \end{bmatrix} = \begin{bmatrix} C_4 & C_4 & C_4 & C_4 & C_4 & C_4 & C_4 & C_4 \\ C_1 & C_3 & C_5 & C_7 & -C_7 & -C_5 & -C_3 & -C_1 \\ C_2 & C_6 & -C_6 & -C_2 & -C_2 & -C_6 & C_6 & C_2 \\ C_3 & -C_7 & -C_1 & -C_5 & C_5 & C_1 & C_7 & -C_3 \\ C_4 & -C_4 & -C_4 & C_4 & C_4 & -C_4 & -C_4 & C_4 \\ C_5 & -C_1 & C_7 & C_3 & -C_3 & -C_7 & C_1 & -C_5 \\ C_6 & -C_2 & C_2 & -C_6 & -C_6 & C_2 & -C_2 & C_6 \\ C_7 & -C_5 & C_3 & -C_1 & C_1 & -C_3 & C_5 & -C_7 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix} \tag{2.11}$$

can be computed as

$$\begin{bmatrix} y(0) \\ y(2) \\ y(4) \\ y(6) \end{bmatrix} = \begin{bmatrix} C_4 & C_4 & C_4 & C_4 \\ C_2 & C_6 & -C_6 & -C_2 \\ C_4 & -C_4 & -C_4 & C_4 \\ C_6 & -C_2 & C_2 & -C_6 \end{bmatrix} \begin{bmatrix} x(0)+x(7) \\ x(1)+x(6) \\ x(2)+x(5) \\ x(3)+x(4) \end{bmatrix} \tag{2.12}$$

and

$$\begin{bmatrix} y(1) \\ y(3) \\ y(5) \\ y(7) \end{bmatrix} = \begin{bmatrix} C_1 & C_3 & C_5 & C_7 \\ C_3 & -C_7 & -C_1 & -C_5 \\ C_5 & -C_1 & C_7 & C_3 \\ C_7 & -C_5 & C_3 & -C_1 \end{bmatrix} \begin{bmatrix} x(0)-x(7) \\ x(1)-x(6) \\ x(2)-x(5) \\ x(3)-x(4) \end{bmatrix} \tag{2.13}$$

This decomposition is possible because the matrix of cosine constants, Γ, is symmetric in the horizontal direction. In general,

$$\Gamma_{m,n} = \Gamma_{m,N-1-n} \ , \quad for\ m\ even$$

and

$$\Gamma_{m,n} = -\Gamma_{m,N-1-n} \ , \quad for\ m\ odd$$

Therefore, for the case of an 8-point DCT,

$$\Gamma_{m,0}\ x(0) + \Gamma_{m,7}\ x(7) = \Gamma_{m,0}\ [x(0) + x(7)], \quad for\ m\ even \tag{2.14}$$

and

$$\Gamma_{m,0}\ x(0) + \Gamma_{m,7}\ x(7) = \Gamma_{m,0}\ [x(0) - x(7)], \quad for\ m\ odd \tag{2.15}$$

This symmetry can also be used to decompose the IDCT. An 8-point 1-D IDCT

$$\begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix} = \begin{bmatrix} C_4 & C_1 & C_2 & C_3 & C_4 & C_5 & C_6 & C_7 \\ C_4 & C_3 & C_6 & -C_7 & -C_4 & -C_1 & -C_2 & -C_5 \\ C_4 & C_5 & -C_6 & -C_1 & -C_4 & C_7 & C_2 & C_3 \\ C_4 & C_7 & -C_2 & -C_5 & C_4 & C_3 & -C_6 & -C_1 \\ C_4 & -C_7 & -C_2 & C_5 & C_4 & -C_3 & -C_6 & C_1 \\ C_4 & -C_5 & -C_6 & C_1 & -C_4 & -C_7 & C_2 & -C_3 \\ C_4 & -C_3 & C_6 & C_7 & -C_4 & C_1 & -C_2 & C_5 \\ C_4 & -C_1 & C_2 & -C_3 & C_4 & -C_5 & C_6 & -C_7 \end{bmatrix} \begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \\ y(4) \\ y(5) \\ y(6) \\ y(7) \end{bmatrix} \tag{2.16}$$

can be computed as

$$\begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} = \begin{bmatrix} v(0) \\ v(1) \\ v(2) \\ v(3) \end{bmatrix} + \begin{bmatrix} v(4) \\ v(5) \\ v(6) \\ v(7) \end{bmatrix} \tag{2.17}$$

and

$$\begin{bmatrix} x(7) \\ x(6) \\ x(5) \\ x(4) \end{bmatrix} = \begin{bmatrix} v(0) \\ v(1) \\ v(2) \\ v(3) \end{bmatrix} - \begin{bmatrix} v(4) \\ v(5) \\ v(6) \\ v(7) \end{bmatrix} \tag{2.18}$$

where

$$\begin{bmatrix} v(0) \\ v(1) \\ v(2) \\ v(3) \end{bmatrix} = \begin{bmatrix} C_4 & C_2 & C_4 & C_6 \\ C_4 & -C_6 & -C_4 & C_2 \\ C_4 & -C_2 & C_4 & -C_6 \\ C_4 & C_6 & -C_4 & -C_2 \end{bmatrix} \begin{bmatrix} y(0) \\ y(2) \\ y(4) \\ y(6) \end{bmatrix} \tag{2.19}$$

and

$$\begin{bmatrix} v(4) \\ v(5) \\ v(6) \\ v(7) \end{bmatrix} = \begin{bmatrix} C_1 & C_3 & C_5 & C_7 \\ C_3 & -C_7 & -C_1 & -C_5 \\ C_5 & -C_1 & C_7 & C_3 \\ C_7 & -C_5 & C_3 & -C_1 \end{bmatrix} \begin{bmatrix} y(1) \\ y(3) \\ y(5) \\ y(7) \end{bmatrix} \tag{2.20}$$

This technique requires more multiplications than other flowgraph algorithms, but it is also has a more regular structure.

Distributed Arithmetic

Distributed arithmetic (DA) was proposed by Peled and Liu [22] as a bit-serial technique for computing the cross product of two vectors without multipliers if one of the vectors is constant. While this technique was originally proposed for use with digital filters, Sun [30] showed that it could be applied to a 1-D DCT or IDCT. To illustrate, consider the vector multiply used to obtain y(1), one point of a 4-point DCT:

$$y(1) = \begin{bmatrix} C_1 & C_3 & -C_3 & -C_1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$

The column vector of inputs, x(0) through x(3), can be rewritten as a matrix of bits in which the n^{th} row is a vector of 1's and 0's expressing x(n) as a two's complement number. If B bits are used,

$$y(1) = \begin{bmatrix} C_1 & C_3 & -C_3 & -C_1 \end{bmatrix} \begin{bmatrix} x_{B-1}(0) & x_{B-2}(0) & \dots & x_1(0) & x_0(0) \\ x_{B-1}(1) & x_{B-2}(1) & \dots & x_1(1) & x_0(1) \\ x_{B-1}(2) & x_{B-2}(2) & \dots & x_1(2) & x_0(2) \\ x_{B-1}(3) & x_{B-2}(3) & \dots & x_1(3) & x_0(3) \end{bmatrix}$$

where $x_i(j)$ is the i^{th} bit of x(j). In a circuit using a conventional multiplier, the cross product would be calculated by multiplying each element of the C vector by

the corresponding element in the x vector and accumulating all of the products. In a DA circuit, each column in the bit matrix is taken one at a time and each bit in the column is multiplied by the corresponding element in C so that

$$\begin{array}{rcccccc} y(1) = & 2^0 & [C_1x_0(0) + & C_3x_0(1) - & C_3x_0(2) - & C_1x_0(3)] \\ & + 2^1 & [C_1x_1(0) + & C_3x_1(1) - & C_3x_1(2) - & C_1x_1(3)] \\ & & \vdots & \vdots & \vdots & \vdots \\ & + 2^{B-2} & [C_1x_{B-2}(0) + & C_3x_{B-2}(1) - & C_3x_{B-2}(2) - & C_1x_{B-2}(3)] \\ & - 2^{B-1} & [C_1x_{B-1}(0) + & C_3x_{B-1}(1) - & C_3x_{B-1}(2) - & C_1x_{B-1}(3)] \end{array}$$

The last line in this equation is subtracted because bit B-1 is the sign bit. Since each column in the bit matrix consists of 4 bits, the product of the column with C has only 2^4 possible values. These values can be stored in a lookup memory.

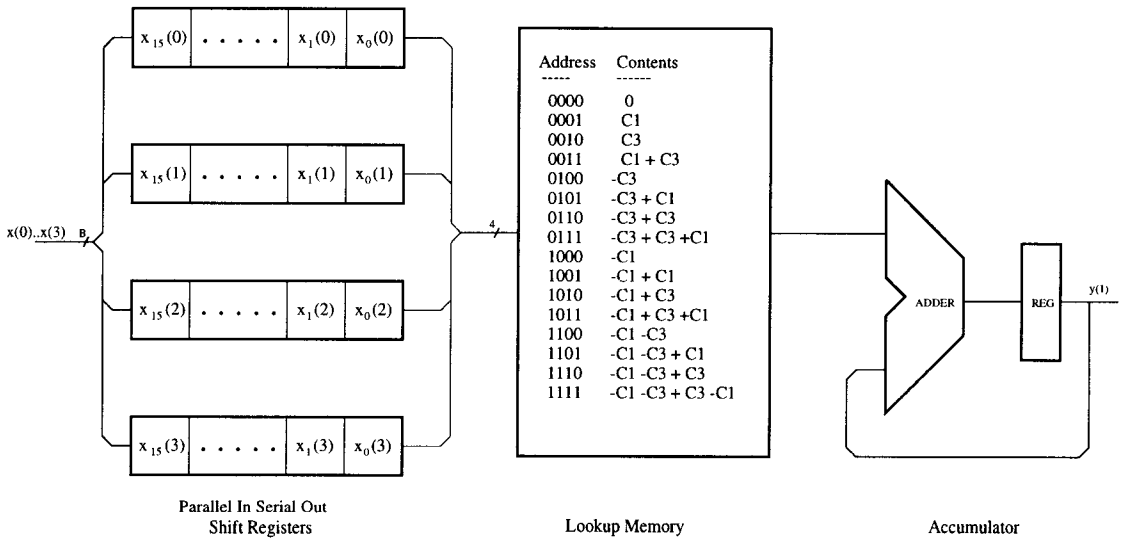


Figure 2.1: Circuit to Compute One Point of a 4-Point DCT Using DA

Figure 2.1 shows a block diagram of a circuit which calculates y(1) using this technique. The circuit consists of four parallel-in serial-out (PISO) shift registers, a lookup table memory and an accumulator. To calculate y(1), the pixel inputs, x(0) through x(3), are written into the PISO shift registers. These inputs are clocked out of the shift registers one bit at a time, starting with the least significant bit (LSB). The shift register outputs form the address input of the lookup memory. On each

clock cycle, one bit is shifted out of the shift registers and a new value comes out of the lookup table. The accumulator output is shifted right by one bit and added to the lookup table output. This right shift accomplishes multiplication by successive powers of two. After the last bit is shifted out and the last lookup value is accumulated, the output of the register holds y(1), the desired DCT coefficient. Since each input is represented by B bits, B accumulation cycles are required. Therefore, calculating a DCT coefficient using DA requires only B additions, as compared to N multiplications and additions for direct implementation using a hardware multiplier.

The circuit in Figure 2.1 calculates one point of the DCT. By adding an additional lookup table, the same circuit can be expanded so that it calculates one point of either the DCT or IDCT. The second lookup table is programmed using constants taken from Γ^T, the matrix of cosine constants for the IDCT. For image compression applications, this means that the same hardware can be used for compression and decompression. All that changes is which lookup table is used.

To summarize the DA technique in more general terms, an N point DCT can be expressed as

$$y(k) = \sum_{n=0}^{N-1} \Gamma_k(n)\ x(n) \tag{2.21}$$

where Γ_k is the k^{th} row vector of the Γ matrix. The elements of x can be expressed in two's complement form using B bits as:

$$x(n) = -x_{B-1}(n)\ 2^{B-1} + \sum_{b=0}^{B-2} x_b(n)2^b \tag{2.22}$$

where $x_b(n)$ denotes the b^{th} bit of $x(n)$. Substituting equation 2.22 into equation 2.21,

$$y(k) = -\sum_{n=0}^{N-1} \Gamma_k(n)x_{B-1}(n)\ + \sum_{n=0}^{N-1} \Gamma_k(n) \sum_{b=0}^{B-2} x_b(n)2^b \tag{2.23}$$

Changing the order of summation in the last term,

$$y(k) = -\sum_{n=0}^{N-1} \Gamma_k(n)x_{B-1}(n) + \sum_{b=0}^{B-2} \left(\sum_{n=0}^{N-1} \Gamma_k(n)x_b(n) \right) 2^b \tag{2.24}$$

Additional background information about DA can be found in [24], [35] and [1].

The circuit of Figure 2.1 calculates one point of a DCT. If another lookup table is added to the circuit, it can calculate one point of a DCT or IDCT, depending on which table is selected. To calculate an 8-point DCT or IDCT, eight such circuits can be used in parallel as shown in Figure 2.2. This circuit requires a total of eight PISO shift registers, 16 lookup tables, and eight accumulators. Each lookup table is programmed using one row vector of the DCT matrix of cosine constants, Γ, or one row vector of the IDCT matrix of cosine constants, Γ^T.

To compute an 8-point transform, the input samples x(0) through x(7) are written into the shift registers and then shifted out one bit at a time. The shift register output forms the address to the lookup memories. On each clock cycle, the lookup memory outputs are accumulated. After B accumulation cycles, the results y(0) through y(7) are available at the output of the accumulators. The input and output nodes in the circuit have been labelled to show the quantities that they hold for a DCT and for an IDCT. For each node, the first value shown is for the DCT and the second value, in parentheses, is for the IDCT.

Since the address formed by the shift register outputs is eight bits wide, each lookup table must contain 256 entries. This highlights one problem with the DA architecture: the size of the lookup memory grows exponentially with the number of points in the transform. Using the DA circuit shown in Figure 2.2 to implement an 8-point 1-D DCT and IDCT requires 16 memories, each having $2^8 = 256$ words.

The memory required for the DCT can be reduced by using the flowgraph decomposition in equations 2.12 and 2.13 to split the 8-point matrix multiplication into two 4-point matrix multiplications as follows:

$$\begin{bmatrix} y(0) \\ y(2) \\ y(4) \\ y(6) \end{bmatrix} = \begin{bmatrix} C_4 & C_4 & C_4 & C_4 \\ C_2 & C_6 & -C_6 & -C_2 \\ C_4 & -C_4 & -C_4 & C_4 \\ C_6 & -C_2 & C_2 & -C_6 \end{bmatrix} \begin{bmatrix} x(0)+x(7) \\ x(1)+x(6) \\ x(2)+x(5) \\ x(3)+x(4) \end{bmatrix}$$

and

$$\begin{bmatrix} y(1) \\ y(3) \\ y(5) \\ y(7) \end{bmatrix} = \begin{bmatrix} C_1 & C_3 & C_5 & C_7 \\ C_3 & C_7 & -C_1 & -C_5 \\ C_5 & -C_1 & -C_7 & C_3 \\ C_7 & -C_5 & C_3 & -C_1 \end{bmatrix} \begin{bmatrix} x(0)-x(7) \\ x(1)-x(6) \\ x(2)-x(5) \\ x(3)-x(4) \end{bmatrix}$$

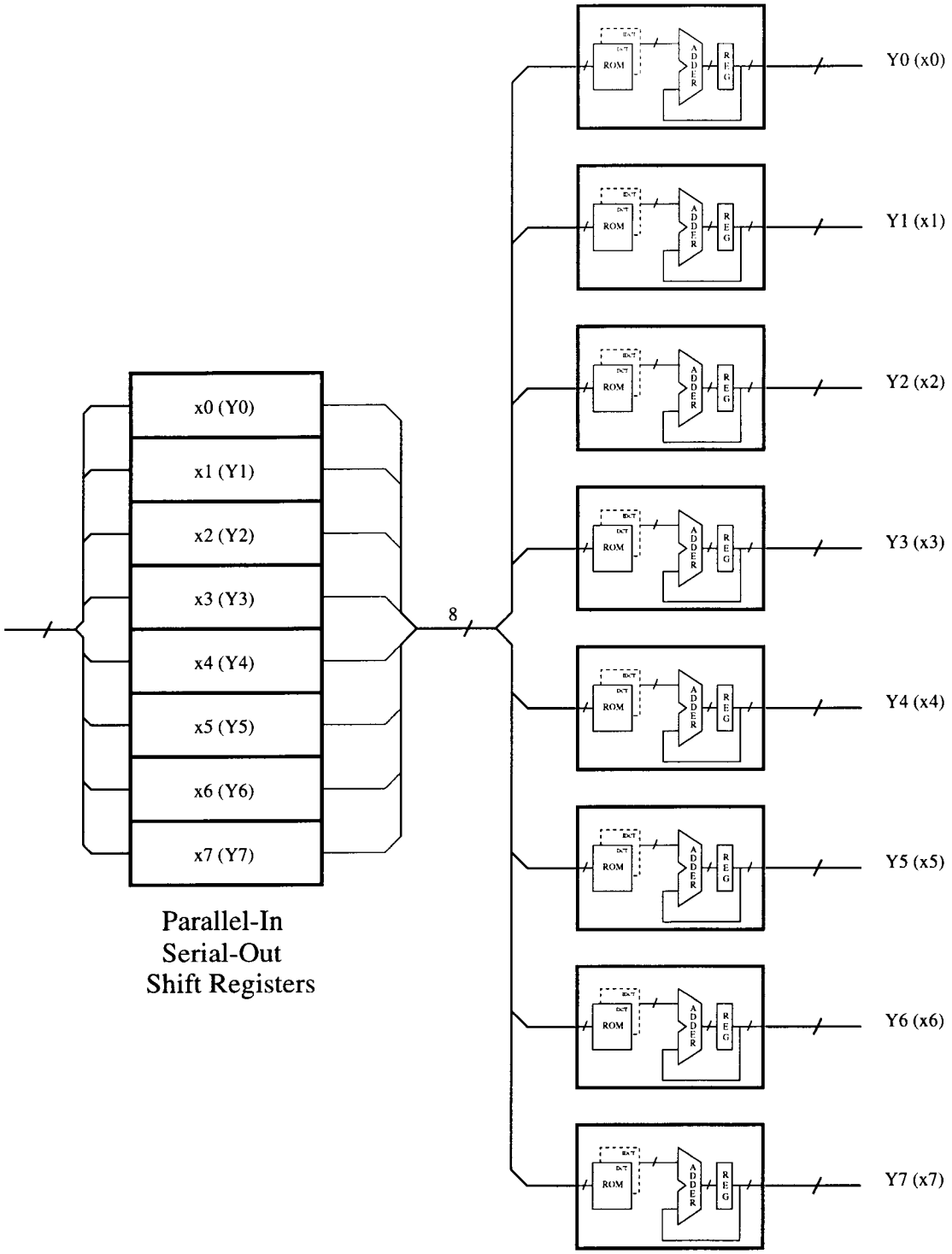


Figure 2.2: 1-D DCT/IDCT Using Pure DA Architecture

Instead of using the pixel inputs directly, these equations use the sums and differences of the pixel inputs. This pairwise sum and difference operation is often referred to as a "butterfly". In similar fashion, the IDCT can also be split into two 4-point matrix multiplications using equations 2.19 and 2.20 as follows:

$$\begin{bmatrix} v(0) \\ v(1) \\ v(2) \\ v(3) \end{bmatrix} = \begin{bmatrix} C_4 & C_2 & C_4 & C_6 \\ C_4 & -C_6 & -C_4 & C_2 \\ C_4 & -C_2 & C_4 & -C_6 \\ C_4 & C_6 & -C_4 & -C_2 \end{bmatrix} \begin{bmatrix} y(0) \\ y(2) \\ y(4) \\ y(6) \end{bmatrix}$$

and

$$\begin{bmatrix} v(4) \\ v(5) \\ v(6) \\ v(7) \end{bmatrix} = \begin{bmatrix} C_1 & C_3 & C_5 & C_7 \\ C_3 & C_7 & -C_1 & -C_5 \\ C_5 & -C_1 & -C_7 & C_3 \\ C_7 & -C_5 & C_3 & -C_1 \end{bmatrix} \begin{bmatrix} y(1) \\ y(3) \\ y(5) \\ y(7) \end{bmatrix}$$

Instead of producing the pixel outputs x(0) through x(7), these equations produce v(0) through v(7) which can be butterflied to form the pixel outputs as shown in equations 2.17 and 2.18.

A DA circuit which uses these decompositions to reduce the size of the lookup tables is shown in Figure 2.3. It will be referred to as a hybrid DA architecture because it uses a flowgraph decomposition together with DA. It is similar to the pure DA architecture in Figure 2.2 with a few differences. For the DCT, it requires that the pixel inputs be butterflied before they are written to the PISO shift registers. For the IDCT, it produces v(0) through v(7) which must be butterflied to produce x(0) through x(7). Most importantly, the addresses to the lookup tables are now four bits wide instead of eight. Therefore, the total memory has been reduced from 16 lookup tables each having 256 words to 16 lookup tables each having 16 words.

Several DA DCT implementations have been reported. Sun [30] was the first to demonstrate the use of DA for the DCT. His implementation was identical to Figure 2.3. Shortly thereafter, Chen and Sun [3] showed a practical VLSI realization of a 16 point DCT using the hybrid DA architecture. Masaki [21] extended the basic DA architecture to high throughput applications by showing that carry-save adders could be used to parallel the additions performed by the accumulator.

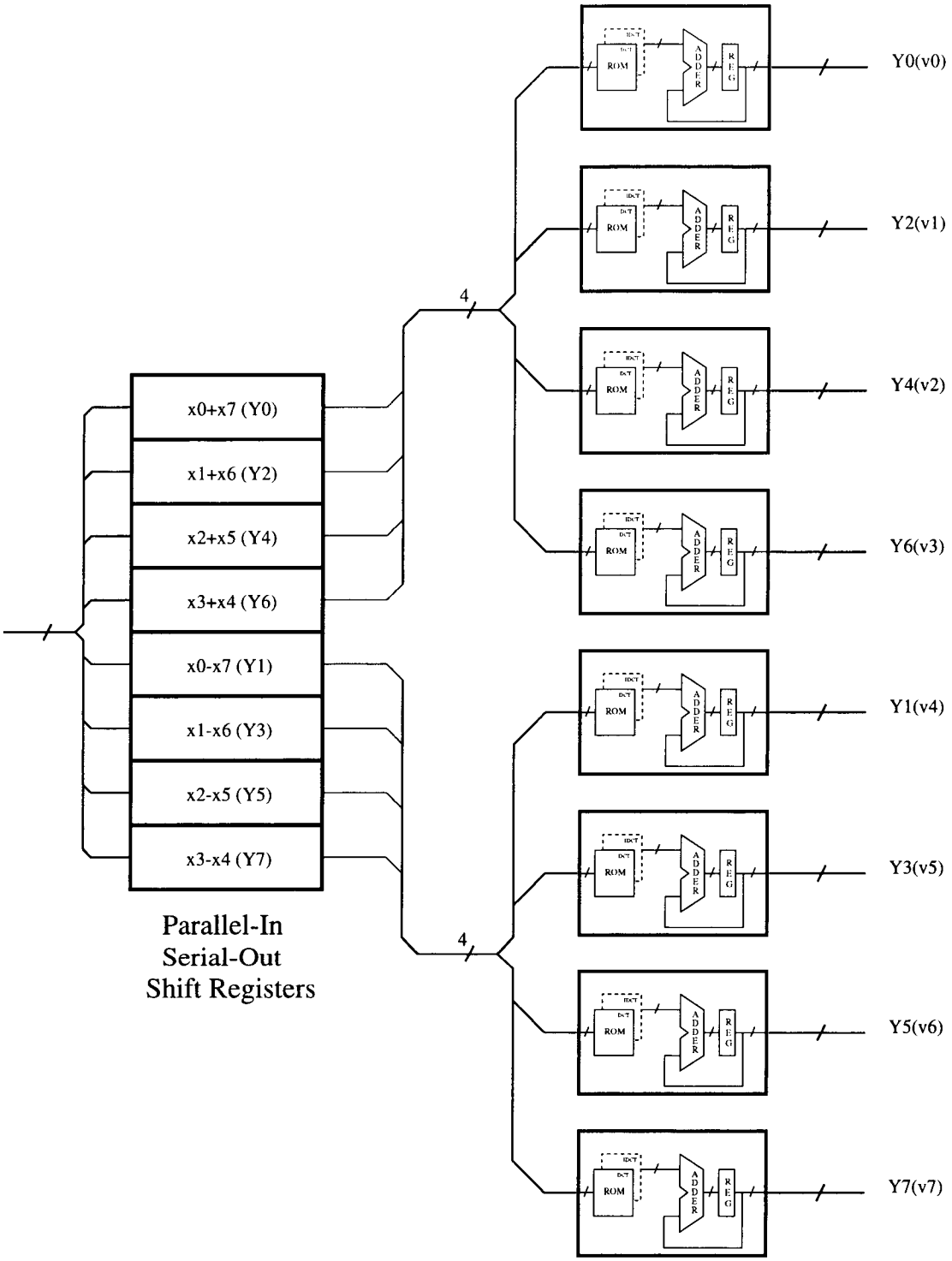


Figure 2.3: 1-D DCT/IDCT Using Hybrid DA Architecture

Other Algorithms

A number of other algorithms for computing the DCT have been proposed. All of the flowgraph algorithms cited so far decompose the 1-D DCT and IDCT and then use the row/column decomposition to realize the 2-D transform using 1-D transforms. Another approach, proposed by Cho [4], is to perform a flowgraph decomposition on the 2-D DCT and IDCT directly. Rather than work with only one row or column of the input matrix at a time, this algorithm uses all of the N^2 inputs in parallel. It has the fewest total multiplications of any published algorithm. For an 8x8 2-D DCT for example, Cho's algorithm requires 96 multiplications and 466 additions, compared to 160 multiplications and 416 additions for Cismas's 1-D flowgraph algorithm.

Another algorithm for computing the 1-D DCT was proposed by Sheu [27]. Like DA, this algorithm is based on using lookup memories. Sheu noted that there are eight unique constants in the 1-D DCT cosine matrix. The matrix is first scaled so that one of the constants is one. All eight constants appear in each row of the matrix, but in a different order on each row. Sheu computes the 8 points of the DCT with seven lookup memories and a series of multiplexers which implement a permutation matrix. Each lookup memory effectively implements multiplication by one of the constants.

An important characteristic of Sheu's algorithm is that the length of each of the lookup memories is 2^B, where B is the number of bits in the input. Sheu's paper considers only the DCT case and claims that the DCT inputs are 8 bits wide. This is true for the first pass of a 2-D DCT. But an important characteristic of the row/column decomposition is that the intermediate results must be stored with greater precision than the inputs and outputs of the 2-D transform. Typically they are stored using 14-16 bits. During the second pass, the DCT and IDCT must therefore accept input which is 14-16 bits wide. Sheu's paper ignores this fact.

Summary and Choice of Algorithm

Table 2.2 summarizes the complexity of several of the algorithms mentioned in this chapter for the case of an 8 x 8 2-D DCT. The first two rows of the table show the number of multiplications and additions for each algorithm. The last two rows show the number of ROM's required and the size of each ROM. The first two columns show the direct implementation from the definition of the 2-D DCT and direct implementation of the 1-D DCT together with the row/column decomposition. The next two columns show Cismas's 1-D DCT flowgraph Cho's 2-D DCT flowgraph. The last columns show the DA architectures and Sheu's lookup table method.

Table 2.2: A Comparision of Algorithms for the 8x8 2-D DCT

Algorithm	Direct	Row/ Column	Cismas	Cho	Sheu	DA	Hybrid DA
Multiplies	4096	1024	160	96	0	0	0
Additions	4096	1024	416	466	1024	2048	2176
# of ROM's	0	0	0	0	7	8	8
ROM Size	—	—	—	—	16k x 16	256 x 16	16 x 16

After careful consideration of the design goals, the hybrid DA architecture was chosen for this implementation. The main consideration was to minimize area of the implementation while maintaining a throughput of at least 16 clock cycles per 1-D DCT and 272 clock cycles per 2-D DCT.

Flowgraph algorithms such as Cismas' require a hardware multiplier to accomplish the desired throughput. The allowable propagation delay through the multiplier is only a fraction of a single cycle. Preliminary investigation showed that about 6000 gates would be required to implement a fully testable single-cycle multiplier using the standard cell libraries available for this design. Furthermore, because the DA architecture calculates the DCT with adders instead of multipliers, it is also easier to scale a DA architecture for high throughput. As the cycle time of the clock

is reduced, it is more difficult to perform a multiplication in one cycle than it is to perform an addition.

Cho's algorithm is potentially the fastest of all the algorithms shown. It also minimizes complexity in terms of the number of mathmetical operations. But in a VLSI realization, it would occupy greater area than the hybrid DA architecture. This is true because Cho's algorithm requires circuitry to do a 1-D DCT, as well as circuitry to do a decomposition of the inputs and outputs of the 2-D DCT. The irregularity of the algorithm would also require a larger control circuit.

A potential disadvantage of the hybrid DA architecture compared to Sheu is that the inputs to the DA circuit are applied in bit-serial, word-parallel fashion. This means that all inputs must be clocked into input registers before any of the coefficients can be calculated. In Sheu's method, the inputs are clocked in bit-parallel, word serial. This means the hybrid DA architecture must have an extra set of registers to hold the inputs. Still, the additional area of these registers is not as great as the area occupied by the much larger ROM's in Sheu's algorithm.

IMPROVEMENTS

The design goal of this research was to minimize the area of a DCT/IDCT block while maintaining a throughput of at least 4.25 clock cycles per input sample. The hybrid DA architecture presented in Figure 2.3 was chosen because of its advantages over other algorithms reviewed in the previous chapter. It does not require a single-cycle hardware multiplier as flowgraph algorithms do. It does not require large lookup tables as Sheu's lookup table algorithm does. It also occupies less area than Cho's 2-D decomposition.

Once the hybrid DA architecture was chosen, an effort was made to reduce the area required to implement it compared to previously published implementations. This chapter describes three techniques which were used. First, some required butterfly operations were done using a bit-serial approach. Second, the lookup tables were implemented using logic instead of ROM's. Finally, four of the 16 lookup tables were eliminated by recognizing a pattern of redundancy in the matrices of cosine constants.

Bit-Serial DCT Butterfly

The hybrid DA architecture in Figure 2.3 computes the DCT and IDCT using less memory than a pure DA architecture. This reduction in memory was accomplished using the decomposition shown in equations 2.12 and 2.13. Using this decomposition, an 8 element matrix multiplication was reduced to two 4 element matrix multiplications. This decomposition is possible because the pixel inputs are butterflied before the matrix multiplication.

In Figure 2.3, it was assumed that the sums and differences of the pixel inputs were written into the shift registers. Forming these sums and differences requires a parallel adder/subtracter circuit which was not shown. At a minimum, this circuit would need one 16-bit adder, 16 exclusive-or gates, and one 16-bit register. For the standard-cell methodology used for this design, these items require 328 gates.

Sun [30] showed that the butterfly for the DCT could be done using bit-serial techniques. In the DA architecture shown in Figure 2.3, the inputs to the matrix multiply are written to PISO shift registers and shifted out in bit-serial fashion. This means the addition and subtraction of the pixel pairs can be done as bit-serial operations. Figure 3.3 shows an improved DA architecture with a bit-serial DCT butterfly. The pixel inputs, x(0) through x(7) are written to the PISO shift registers directly. The outputs of the shift registers are applied to four 1-bit adder/subtractor circuits. The outputs of these circuits form the address to the lookup table memories.

In general, the circuitry for a bit-serial operation requires less area than the same operation implemented in parallel. The four bit-serial butterfly circuits shown in Figure 3.3 requires 166 gates, a reduction of 162 gates.

Bit-Serial IDCT Butterfly

To compute the IDCT using the hybrid DA architecture in Figure 2.3, the outputs of the matrix multiply, v(0) through v(7), are butterflied to produce the pixel outputs, x(0) through x(7), using the following equations:

$$\begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} = \begin{bmatrix} v(0) \\ v(1) \\ v(2) \\ v(3) \end{bmatrix} + \begin{bmatrix} v(4) \\ v(5) \\ v(6) \\ v(7) \end{bmatrix} \tag{3.1}$$

and

$$\begin{bmatrix} x(7) \\ x(6) \\ x(5) \\ x(4) \end{bmatrix} = \begin{bmatrix} v(0) \\ v(1) \\ v(2) \\ v(3) \end{bmatrix} - \begin{bmatrix} v(4) \\ v(5) \\ v(6) \\ v(7) \end{bmatrix} \tag{3.2}$$

where v(0) through v(7) are the results of a matrix multiply as shown in equation 2.19 and 2.20. An important difference between the DCT and IDCT calculation is that the butterfly is performed *before* the matrix multiplication for the DCT but *after* the matrix multiplication for the IDCT. This difference is shown in Figure 3.1.

Since the matrix multiplication outputs come out in parallel, the IDCT butterfly cannot be done as a bit-serial operation if the decomposition in equations 3.1 and 3.2 is used. Given the significant area reduction of a bit-serial butterfly, it would

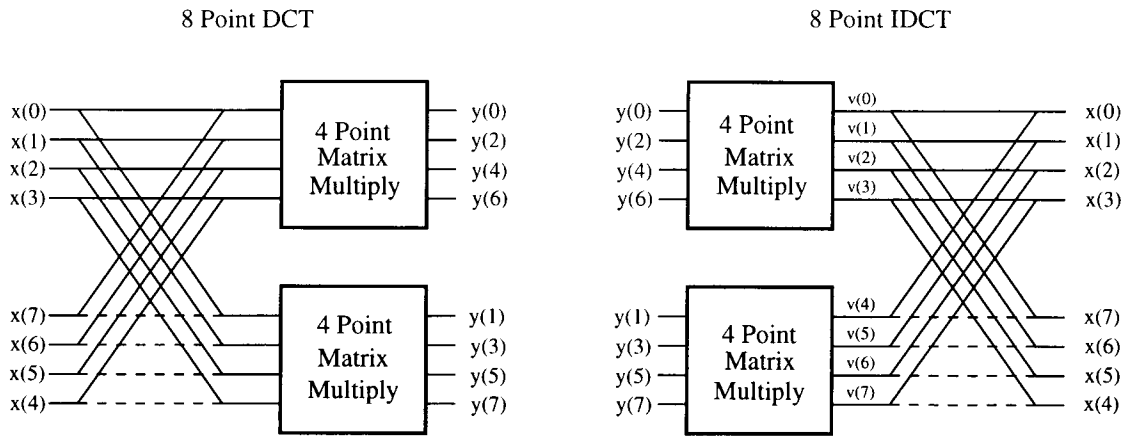


Figure 3.1: Order of Butterfly Operations in 1-D DCT and IDCT

be very useful to find an algorithm for the IDCT which allows the butterfly to be done before the matrix multiplication. Such algorithms have been discovered for the Inverse Discrete Fourier Transform (DFT).

After thorough consideration, our attempts to discover such an algorithm for the IDCT were abandoned. The reason can be seen by comparing the matrix of constants for the IDCT and IDFT. As shown in equation 2.8, the cosine matrix for the IDCT is given by

$$\begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix} = \begin{bmatrix} C_4 & C_1 & C_2 & C_3 & C_4 & C_5 & C_6 & C_7 \\ C_4 & C_3 & C_6 & -C_7 & -C_4 & -C_1 & -C_2 & -C_5 \\ C_4 & C_5 & -C_6 & -C_1 & -C_4 & C_7 & C_2 & C_3 \\ C_4 & C_7 & -C_2 & -C_5 & C_4 & C_3 & -C_6 & -C_1 \\ C_4 & -C_7 & -C_2 & C_5 & C_4 & -C_3 & -C_6 & C_1 \\ C_4 & -C_5 & -C_6 & C_1 & -C_4 & -C_7 & C_2 & -C_3 \\ C_4 & -C_3 & C_6 & C_7 & -C_4 & C_1 & -C_2 & C_5 \\ C_4 & -C_1 & C_2 & -C_3 & C_4 & -C_5 & C_6 & -C_7 \end{bmatrix} \begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \\ y(4) \\ y(5) \\ y(6) \\ y(7) \end{bmatrix} \tag{3.3}$$

The complex exponential matrix for an 8-point 1-D IDFT is given by

$$\begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix} = \begin{bmatrix} W_0 & W_0 & W_0 & W_0 & W_0 & W_0 & W_0 & W_0 \\ W_0 & W_1 & W_2 & W_3 & -W_0 & -W_1 & -W_2 & -W_3 \\ W_0 & W_2 & W_4 & W_6 & W_0 & W_2 & W_4 & W_6 \\ W_0 & W_3 & W_6 & W_1 & -W_0 & -W_3 & -W_6 & -W_1 \\ W_0 & W_4 & W_0 & W_4 & W_0 & W_4 & W_0 & W_4 \\ W_0 & W_5 & W_2 & W_7 & -W_0 & -W_5 & -W_2 & -W_7 \\ W_0 & W_6 & W_4 & W_2 & W_0 & W_6 & W_4 & W_2 \\ W_0 & W_7 & W_6 & W_5 & -W_0 & -W_7 & -W_6 & -W_5 \end{bmatrix} \begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \\ y(4) \\ y(5) \\ y(6) \\ y(7) \end{bmatrix} \tag{3.4}$$

where W_x denotes $e^{j2\pi x/8}$. It was shown in equations 2.14 and 2.15 that it is possible in the DCT to perform a butterfly before the matrix multiplication because the matrix of cosine constants has horizontal symmetry. It can be seen from equation 3.4 above that the IDFT matrix is also symmetric in the horizontal direction. This means that an algorithm exists which can compute the transform by performing the butterfly before the matrix multiply. Unfortunately, the matrix of cosine constants for the IDCT has symmetry only in the vertical direction. This means that the desired IDCT algorithm does not exist, or at least is not obvious and has not yet been found. This can be confirmed by reviewing the papers cited in the previous chapter which discuss flowgraph algorithms. In all of these papers, the last step in the IDCT algorithm is a butterfly operation.

The search for such an IDCT algorithm, or a way to restructure the IDCT to achieve the same effect, is an area for further research. It should be noted that a DA implementation of the DFT and IDFT could use a single bit-serial butterfly circuit.

Replacing ROM with Logic

One disadvantage of the hybrid DA architecture is that many small ROM's are required to implement the lookup tables. For example, the circuit shown in Figure 2.3 requires 16 lookup tables each having 16 words. It is true that the hybrid DA architecture reduced the size of the lookup tables by a factor of $2^{N/2}$ compared to a pure DA architecture. Unfortunately, in a VLSI realization, this does not reduce area by a factor of $2^{N/2}$ because each ROM has overhead associated with input and

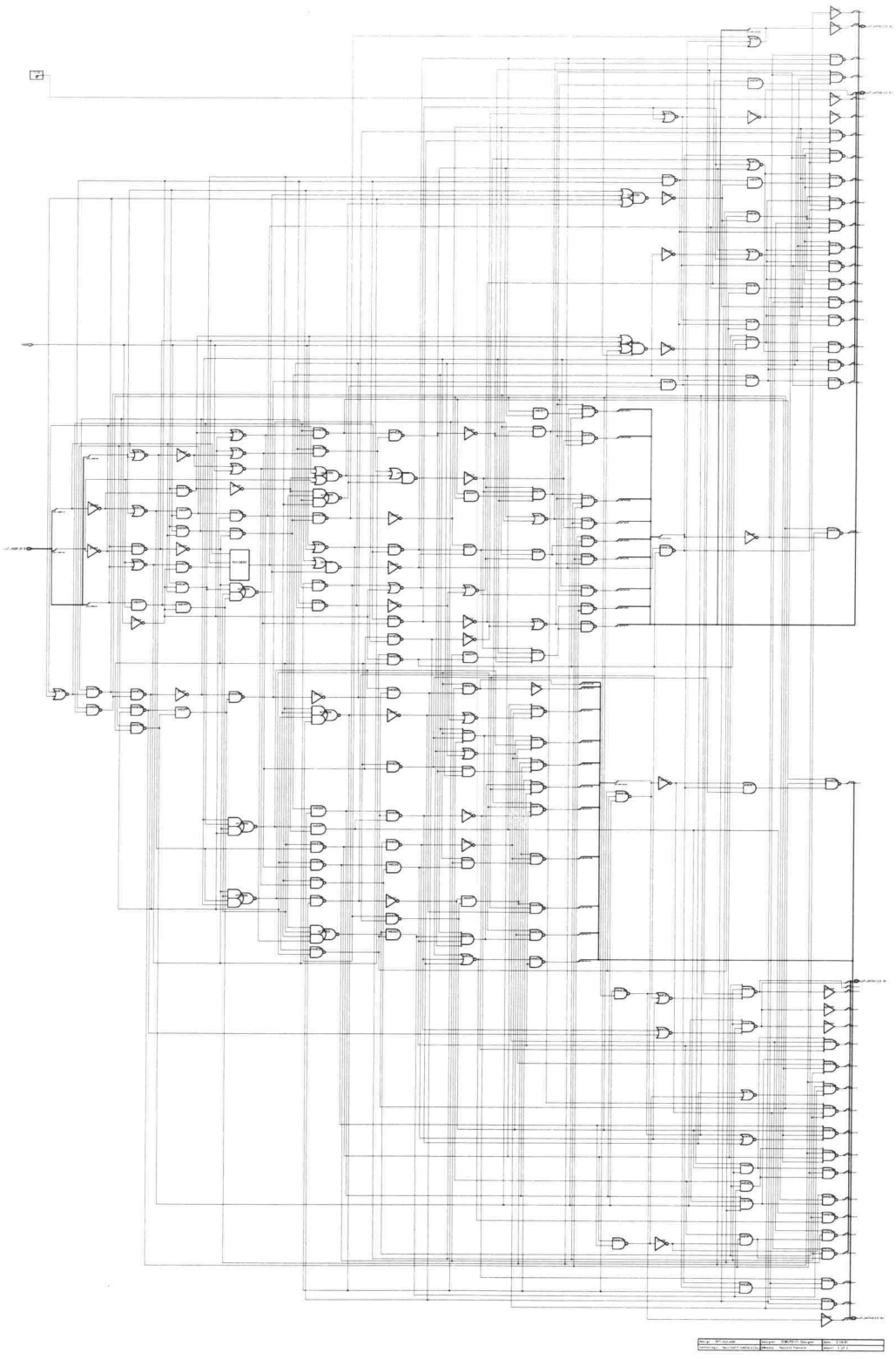


Figure 3.2: Logic To Implement Eight Cosine Lookup Tables

output circuitry. As the size of the ROM is reduced, this overhead occupies a larger percentage of the area.

One technique which can reduce the area of small lookup tables is implementing them using logic gates instead of ROM's. While implementing lookup tables in ROM's becomes less efficient as the tables get smaller, implementing them with logic becomes more efficient. For example, it is estimated that the ROM's required to implement all 16 lookup tables shown in Figure 2.3 would require an area equivalent to 492 gates. Implementing the same lookup tables using logic requires 492 gates. Figure 3.2 is a schematic of the logic required to implement eight of the 16 lookup tables needed for the 8-point DCT/IDCT block shown in Figure 2.3.

In addition to reduced area, there is a further advantage to using logic. Using logic, the lookup tables can be designed with the same process that is used to design the remainder of the circuit. With ROM's, a separate design process is generally required.

Eliminating Four Lookup Tables

As shown in equation 2.13, the odd coefficients of the DCT, y(1), y(3), y(5) and y(7), are computed as

$$\begin{bmatrix} y(1) \\ y(3) \\ y(5) \\ y(7) \end{bmatrix} = \begin{bmatrix} C_1 & C_3 & C_5 & C_7 \\ C_3 & -C_7 & -C_1 & -C_5 \\ C_5 & -C_1 & C_7 & C_3 \\ C_7 & -C_5 & C_3 & -C_1 \end{bmatrix} \begin{bmatrix} x(0)-x(7) \\ x(1)-x(6) \\ x(2)-x(5) \\ x(3)-x(4) \end{bmatrix} \tag{3.5}$$

In the IDCT, the odd coefficients are used to calculate v(4)..v(7) as follows

$$\begin{bmatrix} v(4) \\ v(5) \\ v(6) \\ v(7) \end{bmatrix} = \begin{bmatrix} C_1 & C_3 & C_5 & C_7 \\ C_3 & -C_7 & -C_1 & -C_5 \\ C_5 & -C_1 & C_7 & C_3 \\ C_7 & -C_5 & C_3 & -C_1 \end{bmatrix} \begin{bmatrix} y(1) \\ y(3) \\ y(5) \\ y(7) \end{bmatrix} \tag{3.6}$$

where v(4)..v(7) are used to calculate the pixel outputs of the IDCT.

By closely inspecting equations 3.5 and 3.6, a further improvement can be made to the hybrid DA architecture shown in Figure 2.3. It can be seen that the matrix of cosine constants in equation 3.5 is identical to the matrix in equation 3.6. For a DA

implementation, this means that the same lookup tables which are used for the odd coefficients in the DCT can also be used for the IDCT. Given that each row vector in the matrices of cosine constants becomes one lookup table in a DA architecture, this improvement eliminates four lookup tables. The total memory required for an 8-point DCT/IDCT circuit can be reduced from 16 memories each having 16 words to 12 memories each having 16 words.

Summary of Improvements

Figure 3.3 shows a block diagram of a 1-D DA DCT/IDCT circuit which incorporates the improvements described in this chapter. Comparing this to Figure 2.3, there are four major differences:

- A bit-serial butterfly circuit has been added for the DCT
- Four lookup tables have been eliminated
- The 12 remaining tables have been implemented using logic instead of ROM's.
- A circuit has been added to do the IDCT butterfly

The major components of this circuit are the PISO shift register, bit-serial DCT butterfly, lookup tables, accumulators and IDCT butterfly circuit. The PISO shift registers are labelled to show which input goes to each register and the accumulator outputs are labelled to show which output they produce. The first value shown is for the DCT and the second value, in parenthesis, is for the IDCT.

To calculate a 1-D transform, 8 input samples are written into the PISO and clocked out one bit at a time. If a DCT is being performed, the bit serial butterfly generates the sums and differences of the input samples. If an IDCT is being performed, the input samples pass through the bit-serial butterfly unchanged. The output of the bit-serial butterfly forms the address to the lookup tables. After 16 lookup table values have been accumulated, the matrix multiplication results are available at the output of the accumulators. If an IDCT is being performed, the

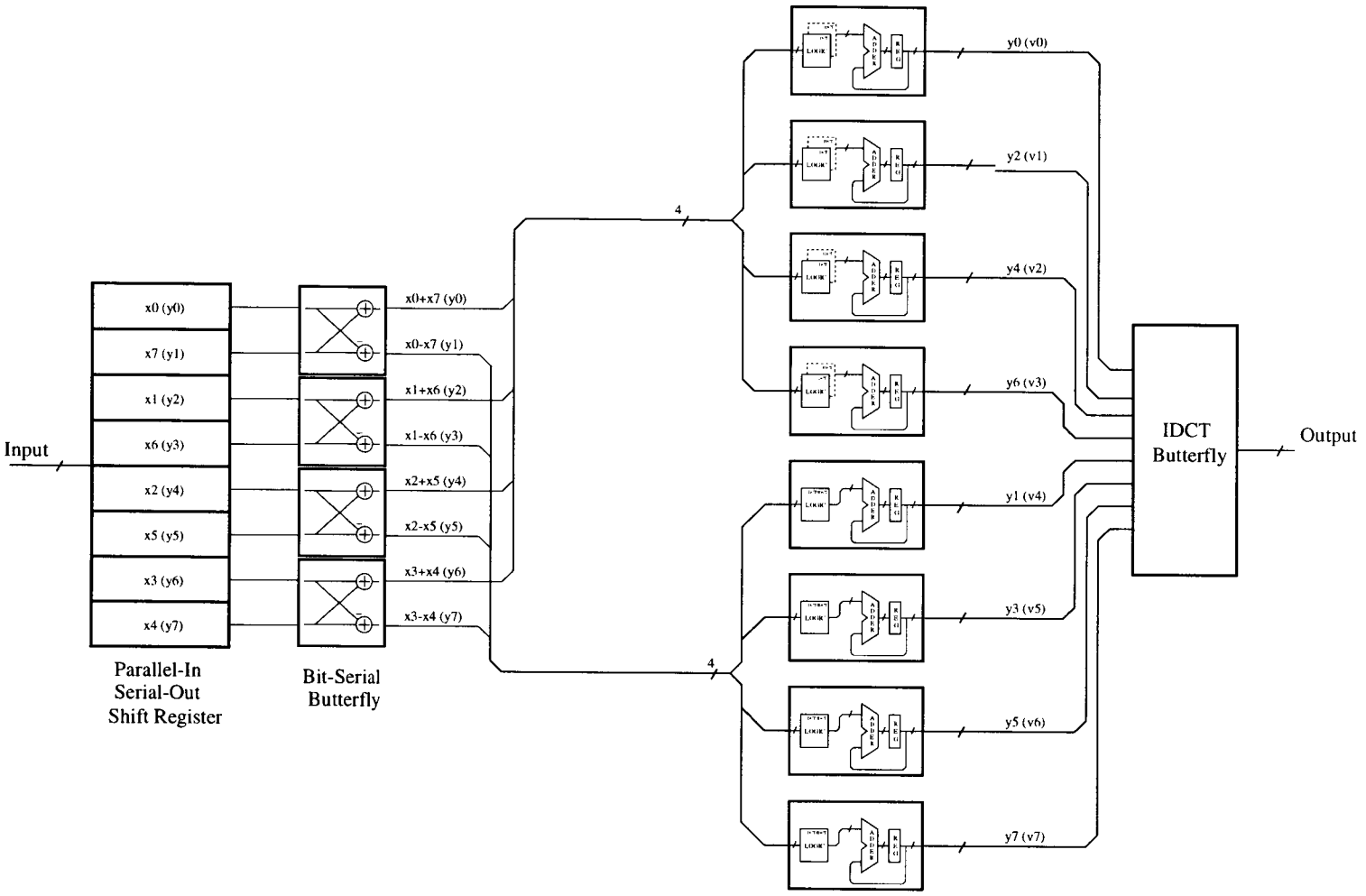


Figure 3.3 - 1D DCT/IDCT Using Improved DA Architecture

v(0) through v(7) values are added and subtracted by the IDCT butterfly circuit to produce the final outputs. If a DCT is being performed, the y(0) through y(7) values pass through the IDCT butterfly unchanged.

IMPLEMENTATION

The previous chapter presented an improved DA architecture for the 1-D DCT and IDCT. This chapter describes a 2-D DCT/IDCT implementation that was done using the improved 1-D DCT/IDCT together with the row/column decomposition. The first step in the implementation was to write a software model in C and perform system level simulations. These simulations verified the accuracy of this architecture and its suitability for JPEG compression. The results of these simulations are described in the following chapter.

After system level verification, a behavioral hardware model was written in the Verilog hardware description language (Verilog HDL). After performing simulations to ensure that the hardware model matched the C mode, the Verilog model was synthesized into a 0.5μm CMOS VLSI circuit using a library of standard-cells.

This chapter describes the details of the hardware implementation. A refinement was made to the row/column decomposition to simplify it. After describing this refinement, the number of bits used to quantize the inputs, outputs and internal nodes of the implementation is explained. Finally, a block diagram of the circuit is presented and each major subsection of the circuit is described.

Row/Column Decomposition

As shown in equation 2.9, a 2-D DCT can be computed using the row/column decomposition as

$$Y = \Gamma \times (X \times \Gamma^T) \tag{4.1}$$

There are two disadvantages to using this equation in a VLSI implementation. First, the X matrix must be multiplied by two different matrices of constants. In VLSI, this means that storage must be allocated for two different sets of lookup tables. A second disadvantage is that the constant matrix is on the left in the first multiplication and on the right in the second multiplication. In a VLSI implementation,

different control logic would be required for the two multiplications. To simplify the implementation, the 2-D DCT can be computed as

$$Y = ((X \times \Gamma^T)^T \times \Gamma^T)^T$$

For image and video processing, the last tranpose operation is customarily omitted, giving

$$Y = (X \times \Gamma^T)^T \times \Gamma^T \tag{4.2}$$

Expressing this calculation in stages,

$$Y_1 = X \times \Gamma^T$$

which is equivalent to performing 1-D DCT's on the rows of X,

$$Y_2 = Y_1^T$$

and

$$Y = Y_2 \times \Gamma^T$$

which is equivalent to performing 1-D DCT's on the columns of of Y_1. The 2-D IDCT can also be computed using the same row/column decomposition:

$$X_1 = (Y \times \Gamma)$$

$$X_2 = X_1^T$$

and

$$X = X_2 \times \Gamma$$

Using this method, both matrix multiplications for the 2-D DCT can use the same matrix of constants, Γ^T and both matrix multiplications for the IDCT use the same matrix of constants, Γ. Figure 4.1 shows a block diagram of a circuit which performs a 2-D DCT using this row/column decomposition. The output of the first 1-D DCT block is Y_1. Y_1 gets written into a memory which is specially designed

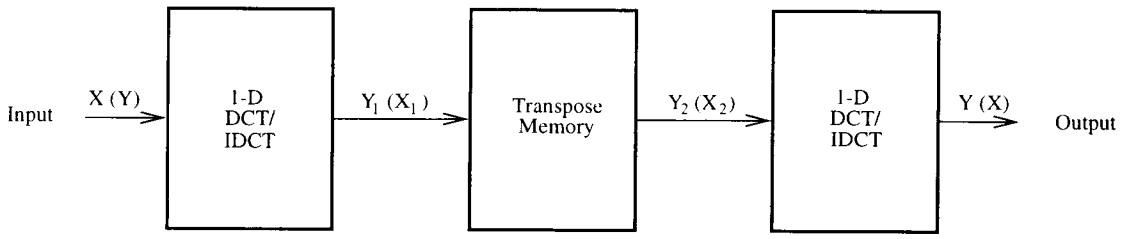


Figure 4.1: Block Diagram of a Row/Column Decomposition Circuit

to transpose data. This transpose memory has 64 locations, logically organized as an 8x8 array. When Y_1 is written to the memory, it is stored in row order. When Y_2 is read from the memory, it is recalled in column order. This switching of the rows and columns is easily accomplished in the addressing logic of the memory. In the last step of the 2-D DCT, Y_2 is transformed to generate Y. The labels in Figure 4.1 show the values at each node for a DCT. The values in parenthesis are for the IDCT.

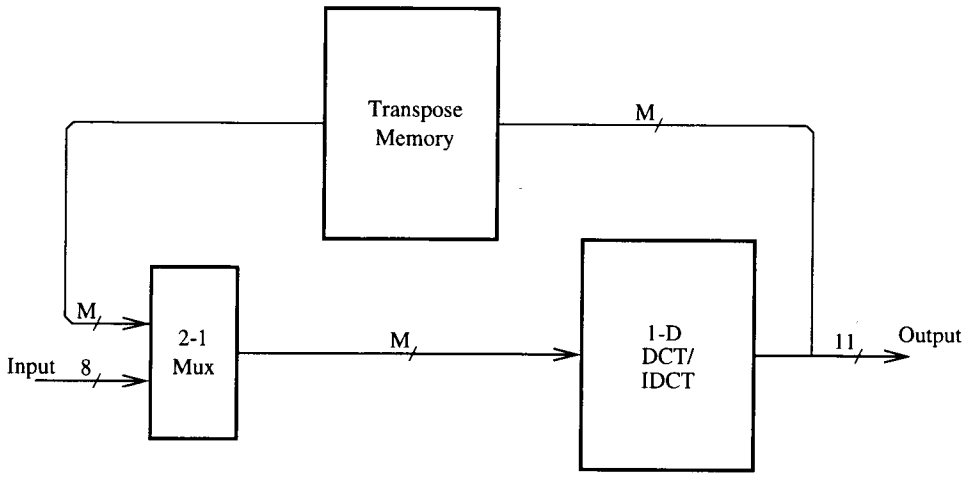


Figure 4.2: Recursive Row/Column Decomposition Circuit

The circuit in Figure 4.1 has two identical 1-D DCT/IDCT circuits. While this architecture would be suitable for very high throughput applications, the throughput goal of this design can be met by a circuit using only one 1-D DCT/IDCT block, as

shown in Figure 4.2. The input to the 1-D DCT/IDCT circuit is multiplexed so that it can come from the external input or from the output of the transpose memory. A 2-D DCT is accomplished in two passes. On the first pass, the 1-D DCT reads X from the input, calculates Y_1 and stores it in the transpose memory. On the second pass, the 1-D DCT reads Y_2 from the memory, calculates Y and writes it to the output.

Quantization

An important issue that must be addressed for the implementation of the circuit is the number of bits used to quantize the input, output and internal nodes of the circuit. The pixels which come in to the 2-D DCT are quantized as 8 bit unsigned integers. This is a standard for digital images which has evolved over time. To minimize the dynamic range of the DC coefficient, the pixel inputs are converted to signed 8-bit integers before processing by the DCT. It is assumed in this hardware implementation that this conversion has already been performed.

Given the number of bits in the input, the number of bits required for the integer part of the 2-D DCT coefficient outputs can be calculated from equation 1.1. The coefficient with the highest possible magnitude is the DC coefficient. For an NxN DCT, the result of the double summation would be N^2 times greater than the 8 bit input. This sum is then scaled by $\frac{2}{N}$ and by $\frac{1}{2}$, a total scaling factor of $\frac{1}{N}$. So the DC coefficient can be $\frac{N^2}{N} = N$ times greater than the input. For N=8, this means that the coefficients require three bits more quantization than the pixels. The integer part of the coefficient must be represented by at least 11 bits. No fractional part is needed for the coefficients because quantization occurs right after the transform. Any fractional part would be removed by quantization. To summarize, the 2-D DCT pixel inputs are 8-bit signed integers and the coefficient outputs are 11-bit signed integers. For the IDCT, the inputs and outputs are reversed.

The internal nodes are all quantized using the same number of bits, shown in Figure 4.2 as M. They are all the same because the 1-D DCT output is the transpose

memory input and the transpose memory output is the 1-D DCT input. The quantization of the node M has two parts: the number of bits to the left of the binary point and the number of bits to the right. Since the internal nodes must be able to handle the 2-D DCT coefficients, the internal nodes must have 11 bits to the left of the binary point. The number of bits to the right of the binary point determines the accuracy of the coefficient outputs. Increasing the number of bits to the right of the binary point increases the accuracy of the coefficients. For this implementation, the internal nodes were quantized with 5 bits to the right of the radix point. This number was derived empirically from system-level simulations presented in the next chapter.

Since this is a fixed-point implementation, all busses in the data path must have 11 bits to the left of the binary point and 5 bits to the right. To accomplish this, the 8 bit pixel inputs for the DCT must be sign-extended to 11 bits. Also, the input samples for the first pass of the DCT and IDCT must be shifted left by 5 bits and padded with 0's to the right of the radix point so that they have the same format as the intermediate results. The quantization of the inputs, internal nodes and outputs for the 2-D DCT can be summarized as

Inputs	Sxx xxxx xxxx	.	00000
Internal Nodes	Sxx xxxx xxxx	.	xxxxx
Outputs	SSS Sxxx xxxx	.	00000

where S represents the sign bit and x's represent data bits.

One-Dimensional DCT

Once the quantization was determined, the 2-D DCT/IDCT circuit in Figure 4.3 was constructed. The transpose memory and input multiplexer are the same as in Figure 4.2. The rest of the circuit performs the 1-D DCT/IDCT. The main parts of the 1-D DCT/IDCT are the input multiplexer, input registers, parallel-in serial-out (PISO) shift-registers, bit-serial butterfly, lookup tables, accumulators, output registers, IDCT butterfly, 8-bit limiter, transpose memory and controller.

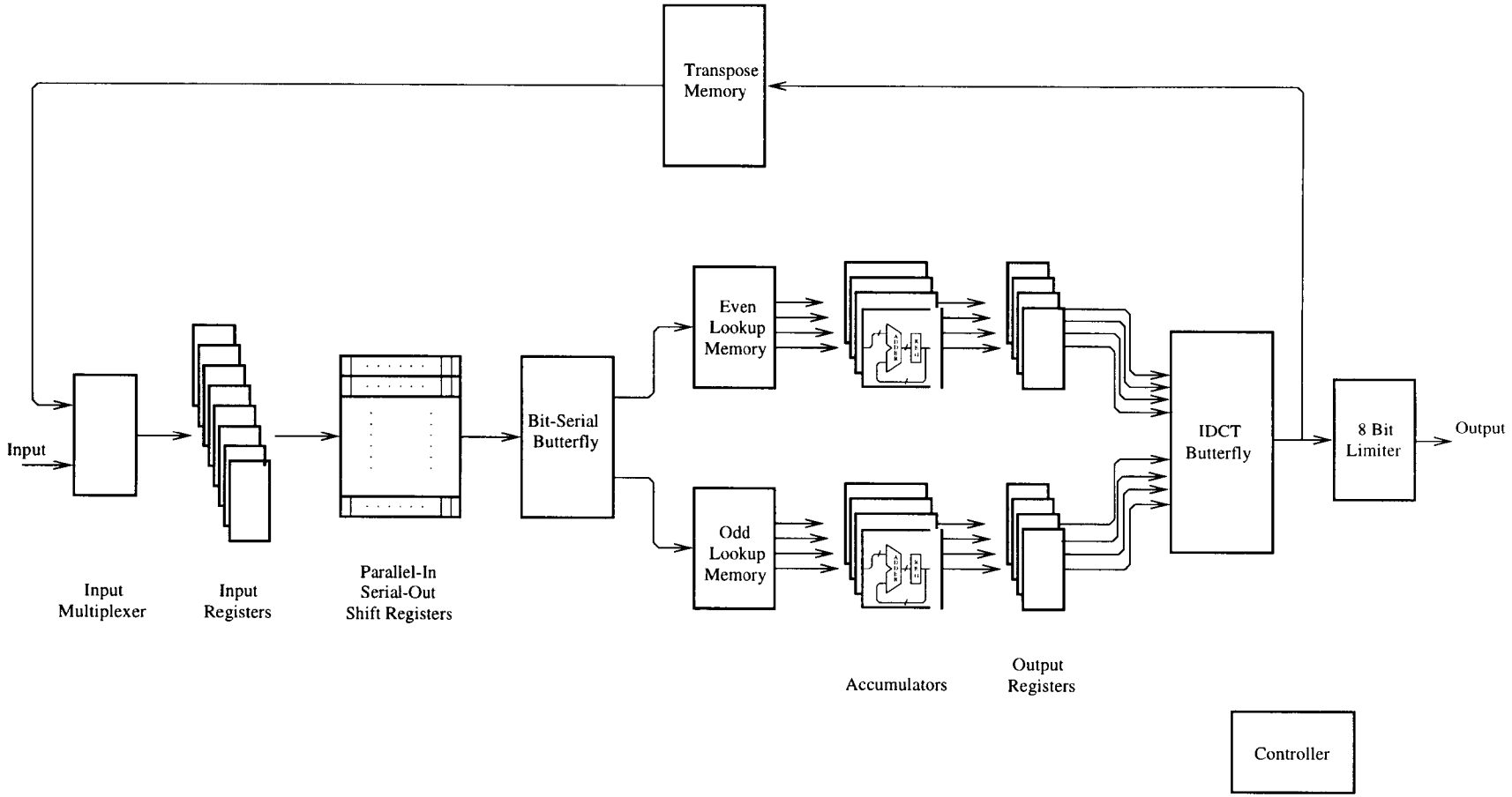


Figure 4.3 - 2-D DCT/IDCT Using Improved DA Architecture

A 1-D transform is calculated every 16 clock cycles. At the start of the 1-D transform, the input samples are clocked from the input registers to the PISO shift registers. These inputs are clocked out of the shift registers one bit at a time, beginning with the LSB. If a DCT is being performed, the bit-serial butterfly produces the sums and differences of the shift register outputs. If an IDCT is being performed, the shift register outputs are passed through the bit-serial butterfly unchanged.

The bit-serial butterfly output is the address for the lookup tables. The even lookup tables produce the values for the even DCT coefficients y(0), y(2), y(4) and y(6). The odd lookup tables produce the values for the odd DCT coefficients y(1), y(3), y(5), y(7). The outputs of the lookup tables are accumulated. After 16 values from the lookup tables are accumulated, the sums are clocked into the output registers and held there until the next set of outputs is available 16 cycles later.

If a DCT is being performed, the output registers hold the coefficients, y(0) through y(7). The IDCT butterfly circuit does not modify these values. It simply multiplexes them one at a time to the output. If an IDCT is being performed, the output registers hold the values v(0) through v(7). These values are butterflied using equations 3.1 and 3.2 to produce the 1-D IDCT outputs x(0) through x(7), which are multiplexed to the IDCT butterfly output one at a time. On the first pass of the 2-D transform, these outputs are written to the transpose memory. On the second pass of the 2-D transform, these outputs pass through the 8-bit limit circuit to the output. The 8-bit limit circuit guarantees that the 2-D IDCT outputs do not exceed 8 bits in magnitude, as explained later in this chapter.

Comparing Figure 4.3 to Figure 3.3, it can be seen that two sets of registers have been added: the input registers and the output registers. These registers were added because of three characteristics of the DA architecture:

- All of the input samples must be available in the PISO registers before the transform calculation can begin

- The input samples must be held throughout the duration of the calculation

- The results of the matrix multiplication all appear at the output on the same cycle.

In Figure 3.3 it was assumed that the inputs were written to the PISO shift registers, requiring 8 cycles. Then the transform calculation was performed, requiring 16 cycles. Finally, the IDCT butterfly multiplexed its inputs to the output one at a time, requiring 8 more cycles. Altogether, this approach would require 32 cycles for a 1-D transform. To speed up the 1-D transform, the input and output registers were added. While one transform is being calculated, the input samples for the next transform can be written to the input registers and the results of the previous transform can be held in the output registers. The means that a 1-D transform can be calculated in 16 cycles.

Figure 4.4 shows how the design is pipelined to compute a 2-D DCT. The diagram shows the contents of the input registers, the PISO shift registers and the output registers as a function of time. Each unit of time in the diagram is 16 clock cycles, the time required to compute a 1-D transform. The 16 cycles starting with clock cycle 0 are shown as S_0, the 16 cycles starting with cycle 16 are shown as S_{16}, etc. The rows of the input matrix X are referred to as R0 through R7.

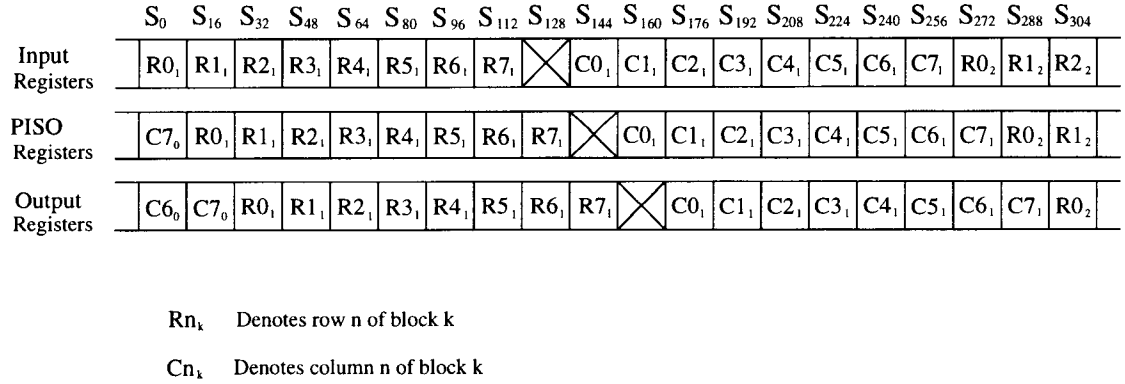


Figure 4.4: Pipeline Diagram for 2-D DCT and IDCT

The first pass of the row/column decomposition lasts from S_0 until S_{144}. During the period starting at S_0, Row 0 is written into the input registers. During the period

starting at S_{16}, Row 0 is clocked into the PISO shift registers and transformed. During the period starting at S_{32}, the transformed Row 0 is held in the output registers. Each element of the row is multiplexed to the output one at a time so that it can be written into the transpose memory.

The second pass of the row/column decomposition lasts from S_{144} until S_{256}. During the period starting at S_{144}, the last row gets written out to the transpose memory. During this same period, the input register starts reading the results from the tranform memory in column order. These columns are referred to in the diagram as C0 through C7.

During the period starting at S_{128} the pipeline is stalled for 16 cycles because the results of the last row transform are not available in time for the first column to be read in. This is not a problem because the 2-D transform still requires only 272 clock cycles to process 64 input samples. Therefore, the design goal of 4.25 clock cycles per pixel is achieved.

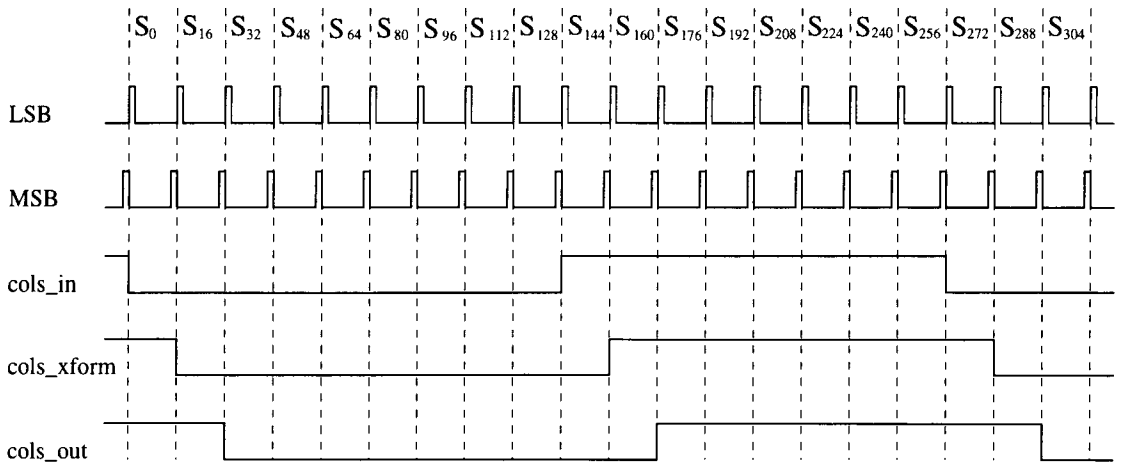


Figure 4.5: Timing Diagram of Control Signals

Controller

Figure 4.5 shows a timing diagram of several important signals generated by the controller. The LSB signal is active for the first cycle of each 1-D transform

calculation. It is used to clear the accumulator outputs at the start of each 1-D transform calculation. The MSB signal is active for the last cycle. The MSB signal causes the contents of the input registers to be transferred to the PISO shift registers. It also causes the accumulator outputs to be transferred to the output registers.

The cols_in and cols_out signals are similar signals which indicate whether the first pass or second pass of the row/column decomposition is being performed. The cols_in signal is active (high) when the inputs for the second pass of the transform are being written to the input registers. When cols_in is low, input comes from the external input. When it is high, input comes from the transpose memory. The cols_out signal is active when the output registers hold results computed during the second pass of the transform.

Two other important control signals are the RST signal and the MODE signal. These are not generated by the controller, but must be set by the external circuitry. The RST signal must be asserted high to initialize the circuit before it is used. The MODE signal determines whether a DCT or IDCT is to be performed. It must be set low for a DCT and high for an IDCT.

The function of each of these control signals will become more clear as each major section of the circuit is described in detail.

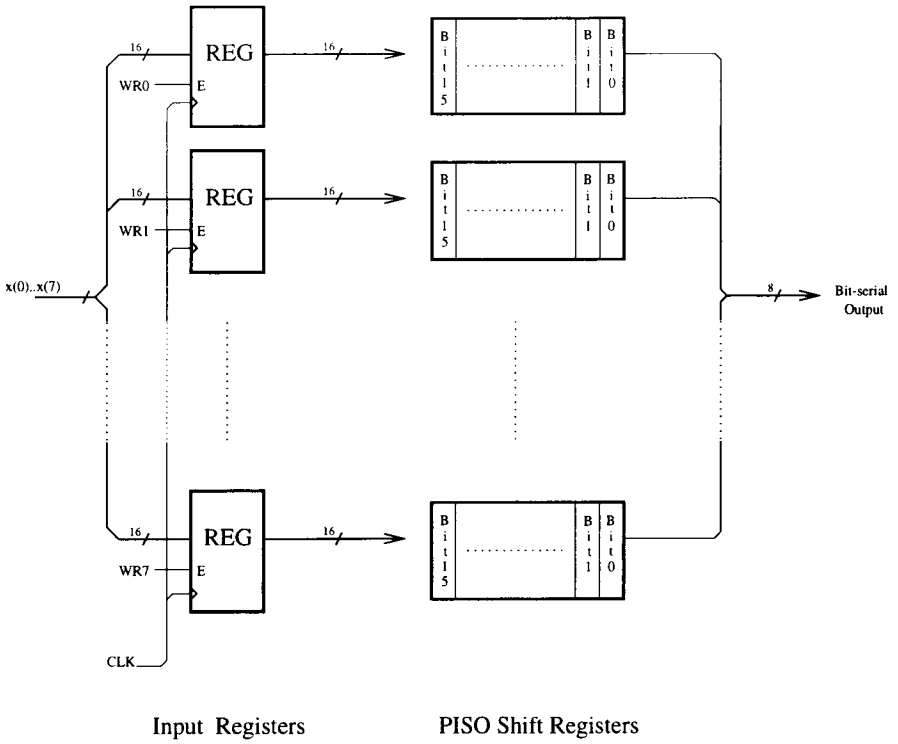


Figure 4.6: Input Registers and PISO Shift Registers

Input Registers and PISO Shift Registers

Figure 4.6 shows a block diagram of the input registers and PISO shift registers. It was noted earlier that all the input samples must be available before a 1-D transform calculation can begin and these samples must be held for the duration of the calculation. To satisfy these requirements, the input samples are written to the input registers. They can be written in any order and at any time before the beginning of a new 1-D transform calculation. At the start of a new transform calculation, the values from the input registers are transferred to the PISO shift registers. On the following clock cycles, the input samples are shifted out one bit at a time beginning with the LSB.

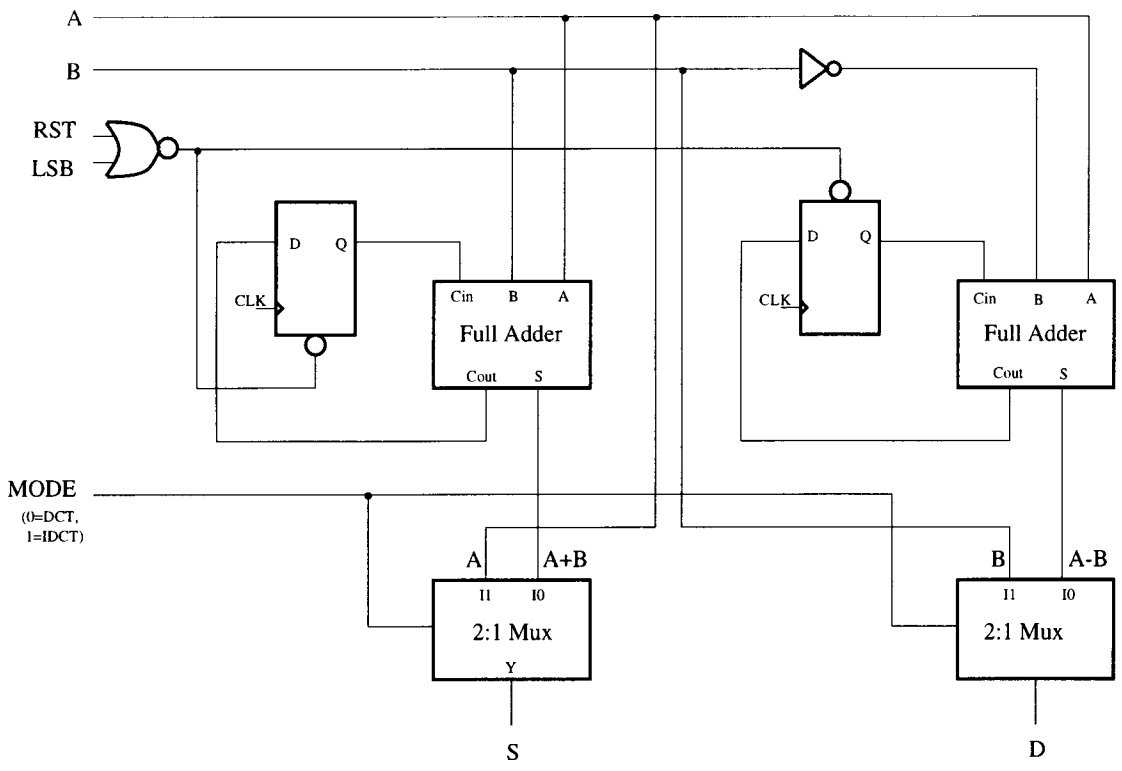


Figure 4.7: Bit Serial DCT Butterfly Circuit

Bit-Serial DCT Butterfly

Figure 4.7 shows the bit-serial butterfly circuit. If the MODE signal indicates that a DCT is to be performed, this circuit generates the sum of the "A" and "B" inputs on the "S" output and the difference of "A" and "B" on the "D" outputs. For the IDCT, the "A" and "B" inputs are passed to the outputs unchanged.

The circuit contains a one bit adder and a one bit subtractor. During cycle 0, the LSB signal is asserted, clearing the carry-in to the adder and setting the borrow-in for the subtractor. On subsequent cycles, the carry and borrow are simply the results of the carry-out and borrow-out from the previous cycle, as stored in the two flip-flips. A total of four such circuits are needed to butterfly the 8-bit output of the bit-serial shift register.

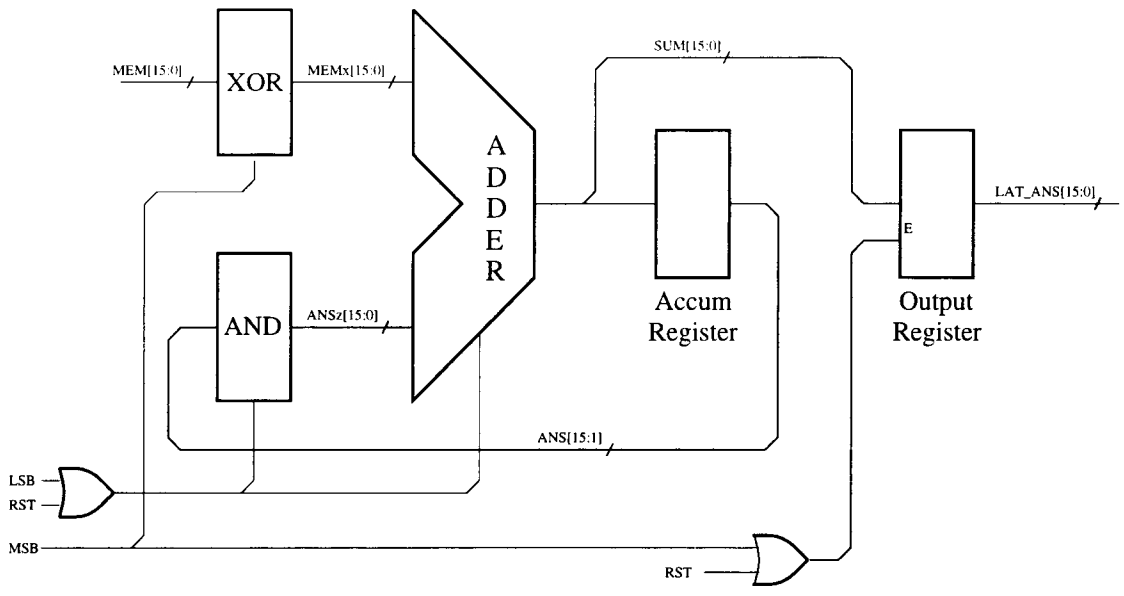


Figure 4.8: Accumulator Circuit

Accumulator

Figure 4.8 shows a block diagram of the accumulator circuit. It accumulates the current input value together with the previous sum shifted right by one bit. For each transform calculation, it accumulates 16 values from the lookup tables in 16 cycles.

As shown in equation 2.24, the 16th cycle represents the sign bit so the lookup table value must be subtracted rather than added during this cycle. The lookup table values are applied to the accumulator on the MEM input. For cycle 0 through cycle 14, the MEM value passes unchanged through the exclusive-or block and is applied to the adder. On cycle 15, the MSB signal is asserted. This causes the one's complement of the MEM value to be applied to the adder and causes the adder's carry in to be asserted. The net result is that the two's complement of the MEM value is added to the accumulated sum, which is equivalent to subtraction.

The accumulated sum is stored in the accumulator register at the end of each cycle. At the end of cycle 15, this sum is clocked into the output register where

it is held for the following 16 cycles. As each new 1-D transform is started, the accumulated sum from the previous transform must be cleared from the accumulator register. This function is accomplished by the AND block which passes zeros to the second adder input when the LSB signal is asserted during cycle 0. During all other cycles, the AND block shifts ANS[15:1] right by one bit and sign extends it to generate ANSz[15:0]. This performs the multiplication by 0.5.

During reset, the MEM value is zero and the LSB signal is also asserted. Since both of the adder's inputs are zero, the accumulator register gets clocked to zero. The output register also gets clocked to zero because it's enable signal is active if RST is asserted.

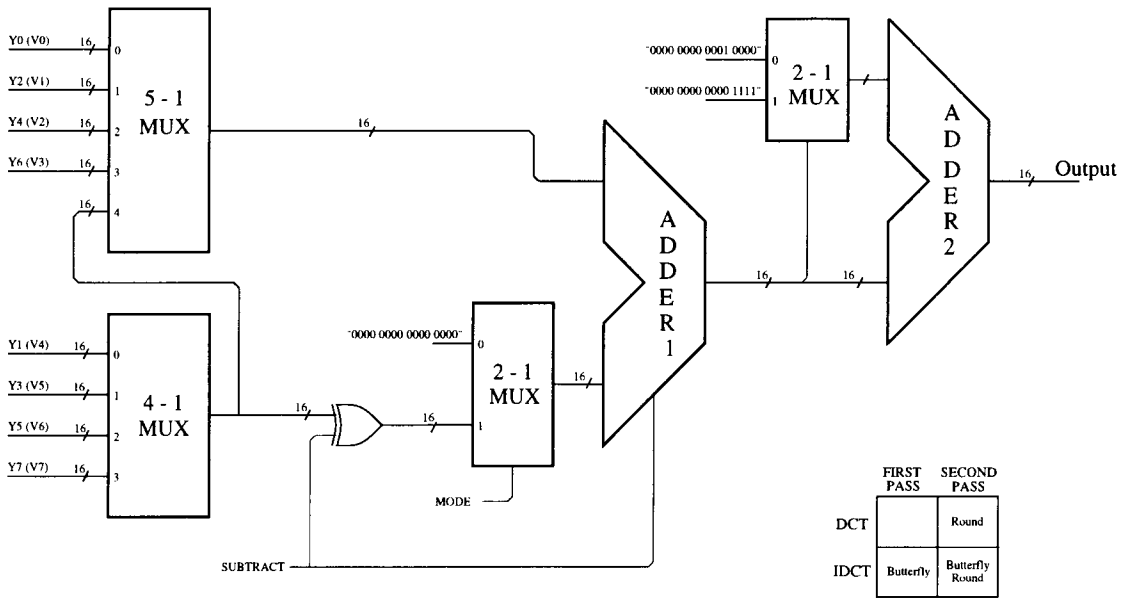


Figure 4.9: IDCT Butterfly Circuit

IDCT Butterfly

At the end of every 16 clock cycles, the results of the matrix multiply get clocked into the output registers and applied to the inputs of the IDCT butterfly circuit. This circuit performs several functions. Since the 1-D transform results are all clocked into the output registers at the same time, this circuit multiplexes them to

the output one at a time. If an IDCT is being performed, this circuit butterflies the v(0) through v(7) terms from the output registers to produce the IDCT outputs as shown in equations 3.1 and 3.2. For the second pass of both the DCT and the IDCT, this circuit rounds the outputs to the nearest integer.

Figure 4.9 shows a block diagram of the IDCT Butterfly Circuit. The table in the lower right corner of this figure summarizes which operations are performed for the first and second passes of the DCT and IDCT.

The circuit contains two adders: ADDER1 for the IDCT butterfly and ADDER2 for rounding. For the IDCT butterfly, one input to ADDER1 comes from the 5:1 multiplexer and the other input comes from the 4:1 multiplexer. The sum of these two inputs is produced first. Then the controller asserts the SUBTRACT signal to produce the difference of these two inputs. If a DCT is being performed, the MODE signal forces the output of the first 2:1 multiplexer to zero. Each of the DCT values y(0) through y(7) are applied to the first adder input one at a time and pass through ADDER1 unchanged.

During the second pass of the 2-D transform, the outputs of ADDER1 must be rounded. To perform unbiased rounding to an integer value, ADDER2 adds 0.5 to positive numbers or 0.5 - 2^{-R} to negative numbers, where R is the number of bits to the right of the binary point. After this addition, the fractional part is truncated to produce the rounded result. The second 2:1 multiplexer generates the appropriate rounding value based on the sign bit of the number to be rounded.

8-Bit Limiter

The last stage of the 2-D DCT/IDCT circuit in Figure 4.3 is the 8-bit limiter. For image and video compression applications, the output of the second pass of the IDCT must be limited to 8 bit signed integers. This limiting is necessary because the DCT coefficients get quantized during compression. When the quantized coefficients are applied to the IDCT during decompression, it is possible for some of the pixels in the output to exceed 8 bits in magnitude.

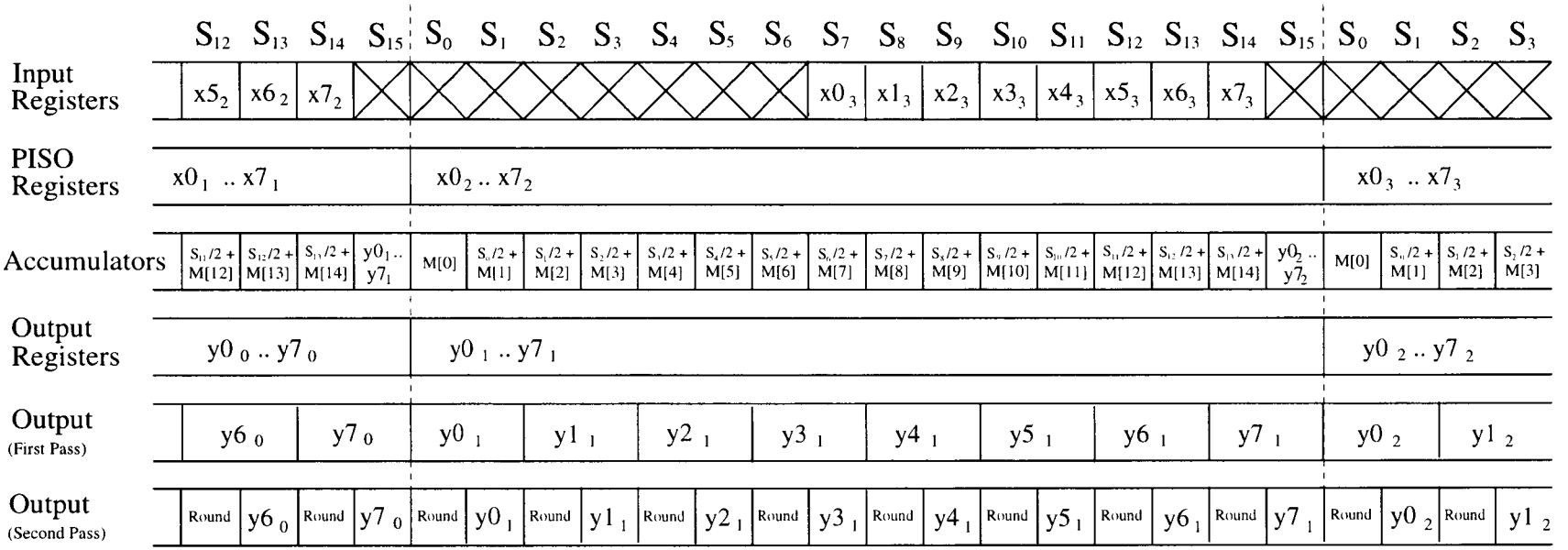
The limiter circuit is enabled if the MODE signal and the cols_out signal indicate that the second pass of an IDCT is being performed. The limiter works by comparing bit 10 (the sign bit) of the integer portion of the adder result to bit 7. If they are not equal, an overflow has occurred and the output is forced to the maximum positive value $(2^7 - 1)$ or to the maximum negative value (-2^7), depending on the sign bit.

Pipelining

Figures 4.10 and 4.11 are timing diagrams for the 1-D DCT and 1-D IDCT which illustrate how the design was pipelined. Each 1-D transform requires 16 cycles which are labelled S_0 through S_{15}. Each row in these diagrams shows the contents of one set of registers as a function of time.

Figure 4.10 shows a total of 4 different sets of inputs moving through the pipeline, labelled with subscripts 0 through 3. The left side of the diagram shows the end of the previous 1-D transform and the right side shows the beginning of the next transform. The center shows all 16 cycles of the current transform. The first row of the diagram shows the input samples being written to the input registers. The second row shows the input samples being shifted out of the PISO registers and used as the inputs to the matrix multiply. The third row shows the lookup memory values being added in the accumulators until the final matrix multiplication results appear at the end of S_{15}. The fourth row shows these results being clocked into the output registers. The fifth and sixth rows show the results being multiplexed to the output during the first and second passes respectively. During the second pass, the answers are rounded as they are multiplexed to the output.

Figure 4.11 shows the corresponding timing diagram for the IDCT. It is very similar to Figure 4.10 except that the results of the matrix multiply which are stored in the output registers are v(0) through v(7). The IDCT outputs are formed by adding and subtracting these terms. Also, the outputs are limited on the second pass of the IDCT.

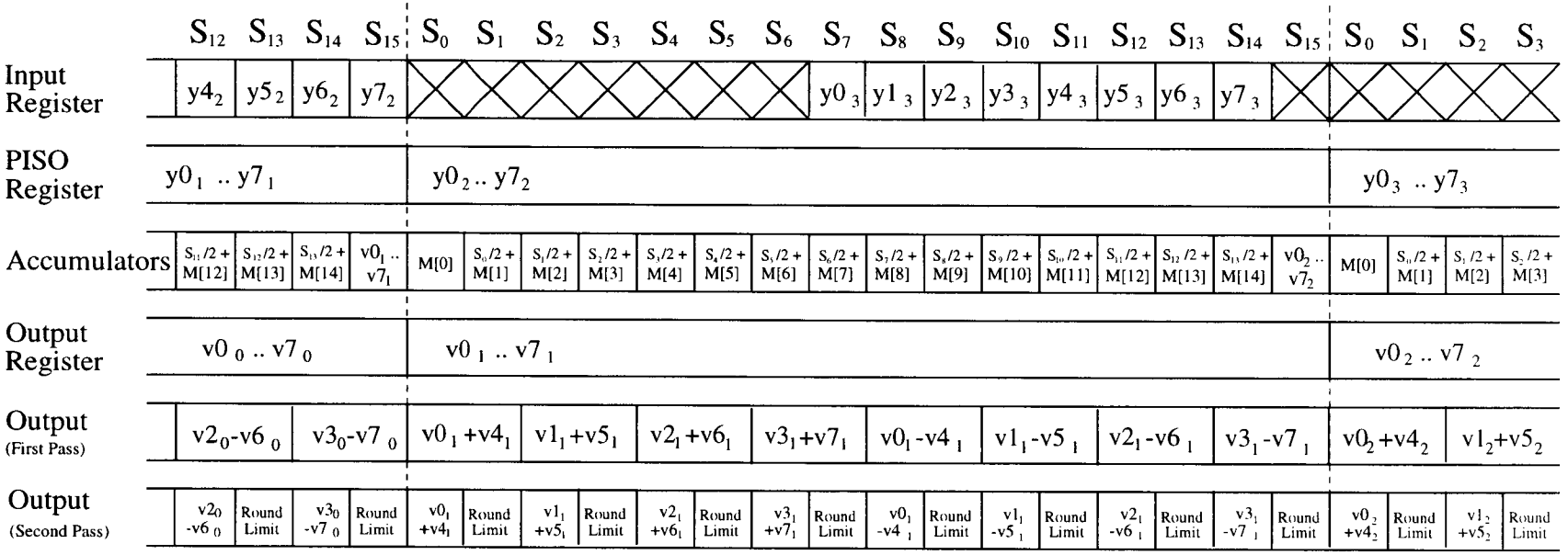


$x7_2$ Denotes pel 7 of the 2nd set of inputs

$y0_2$ Denotes coefficient 0 of 2nd set of DCT outputs

M[I] Denotes output of lookup memory during cycle I

Figure 4.10 - 1-D DCT Pipelining



xI_j Denotes pixel I of the jth set of outputs

yI_j Denotes coefficient I of the jth set of inputs

M[I] Denotes lookup table output during cycle I

Figure 4.11 - 1-D IDCT Pipelining

From these two diagrams, it can be seen that the throughput for this design is 16 cycles per 1-D transform while the latency is 48 cycles. The latency comes from the fact that the DA architecture must have all inputs available before it can begin the transform and does not produce its outputs until the end of the 16th cycle.

RESULTS AND CONCLUSIONS

In order to verify that the architecture presented in Figure 4.3 would be suitable for JPEG compression, simulations were performed using three models of the architecture. A Matlab model was written to verify the feasibility of the design and to get a rough idea of the quantization required. A C language model was written to perform actual JPEG compression. A behavioral hardware model was written in the Verilog Hardware Description Language (HDL). The Verilog model was synthesized into gates to estimate the size of the design and simulated at the gate level to estimate its maximum clock speed. This chapter presents the results of these simulations and discusses the advantages and disadvantages of this architecture relative to a flowgraph implementation.

Accuracy Results

An important issue for fixed-point DSP circuits is the effect of finite word length. Figure 4.2 showed the recursive row-column decomposition used for the improved DA circuit. The number of bits used to quantize the internal nodes was labelled as "M". Increasing M increases the accuracy of the transform results, but it also increases the die area required to implement the design.

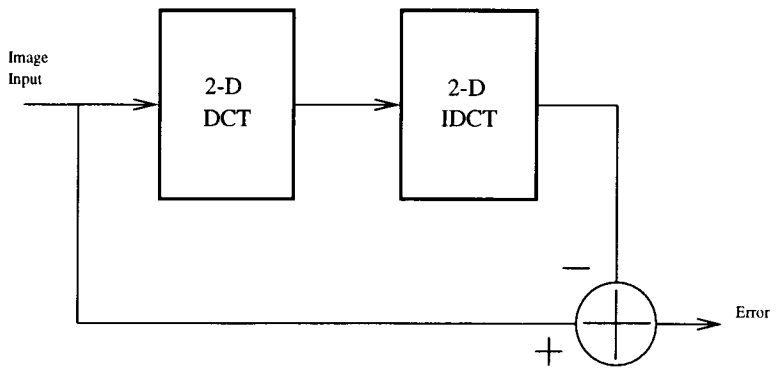


Figure 5.1: Test Setup for Measuring 2-D DCT/IDCT Accuracy

To determine the value of M, models were written in Matlab and in C for the 2-D DCT and IDCT circuits shown in Figure 4.3. These models were used to perform simulations which determine accuracy as a function of M. The basic simulation method is shown in block diagram form in Figure 5.1. 8x8 blocks of image data were compressed using the DCT model and decompressed using the IDCT model. The decompressed results are subtracted from the original data to generate an error term. This method is used by Chen [2].

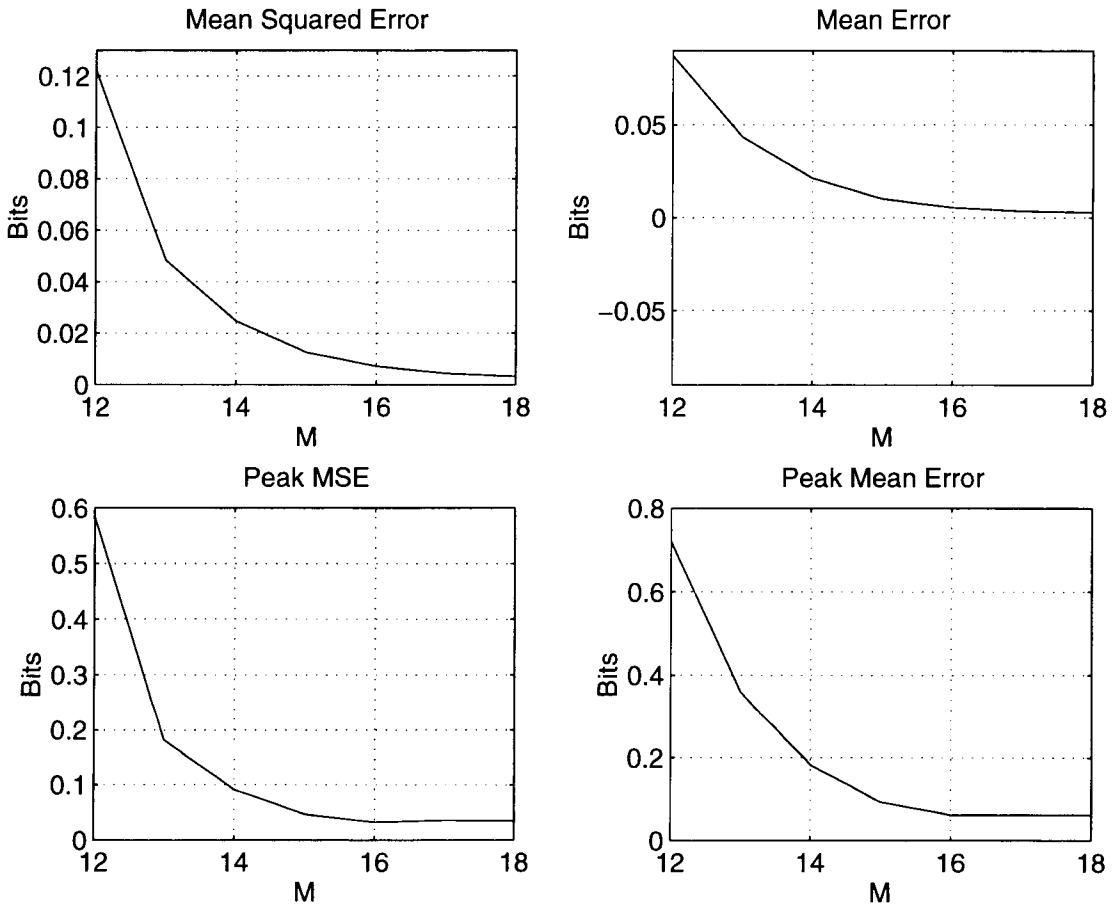


Figure 5.2: Error Statistics for Compressing and Decompressing Random Blocks

The error term generated by this simulation can be analyzed in several ways. One way is to compute the statistics of the error. The first set of simulations used this approach. Using Matlab, a set of 1000 8x8 blocks of random numbers

were generated and processed as shown in Figure 5.1. The random numbers were integers uniformly distributed between -128 and +127. Each block was compressed and decompressed with the 2-D DCT and IDCT models for values of M ranging from 12 to 18 bits. Four error statistics were generated from the simulation: mean error, mean-squared error (MSE), peak mean error and peak MSE. Figure 5.2 shows a graph of the error in bits as a function of M for each of these four statistics. The error statistics were calculated as follows. For the simulation, data was processed as an 8x8 matrix. Mean error and MSE were calculated for each position in the matrix individually. The largest of these individual values were taken as the peak mean error and peak MSE. After these peak values were extracted, the mean error and MSE for all positions in the matrix were averaged to form the total mean error and MSE shown in the graphs. The inputs were 8 bit numbers, so an error of 1 bit is less than 1% of full scale.

These error statistics shown in this figure are defined in an International Telecommunications Union (ITU) standard for DCT/IDCT accuracy in video compression. This standard has also been adopted as IEEE standard 1180. A brief summary appears in [10]. While it isn't necessary for an image compression chip to meet the accuracy limits defined in this standard, the error statistics it defines were useful in getting a rough idea of reasonable values for M. It appears from these graphs that increasing M beyond 16 bits provides a minimal increase in accuracy.

While the information from the Matlab simulation was useful, there was a problem with this model. It is preferable to do a simulation using real image data rather than randomly generated blocks, but the Matlab model was too slow to do an entire image. It only simulated about 100 8x8 blocks per hour. At this rate, it would take about a week to do a VGA color image. To get around these problems, a C language model was written to simulate the 2-D DCT/IDCT circuit shown in Figure 4.3. This C model simulates about 10,000 blocks per minute.

Using this model, an image was read in and divided into 8x8 blocks. Each block was compressed and decompressed with the value of M set to 14, 15 and 16 bits. The same simulations were also run for a 32-bit floating point implementation. The

error was analyzed by forming a histogram showing the percentage of pixels for which the error is ± 1 bit, ± 2 bits, etc. This technique is described by Carlach [1]. Table 5.1 shows the histograms for M = 14, 15 and 16 as well as the floating point implementation. From these histograms it was decided that setting M to 16 bits provides accuracy which is close enough to floating point and that increasing M beyond 16 bits would not increase accuracy significantly.

Table 5.1: Histogram of DCT/IDCT Accuracy for Various Quantization Levels

Bits	-2	-1	0	+1	+2
14	0.000	5.486	90.164	4.346	0.004
15	0.000	3.133	94.296	2.572	0.000
16	0.000	2.500	95.124	2.376	0.000
Floating Point	0.000	2.283	95.388	2.329	0.000

There was another reason to write C models for the DA DCT/IDCT block. C models had been written for the existing DCT/IDCT flowgraph design, including models to simulate every step of JPEG compression and decompression. The C model for the DA architecture was used to perform JPEG compression and decompression on 9 test images. These test images were chosen from a collection of about 60 images which had been simulated using the flowgraph architecture. The test images were chosen because they had the widest distribution of errors for the flowgraph architecture. The decompressed images were viewed to subjectively verify image quality. Also, the compression ratio obtained with the DA architecture was compared to the existing flowgraph architecture. The JPEG files compressed using the DA DCT were the same size as those compressed by the flowgraph DCT to within 0.1%.

Gate Count

In order to estimate the size and maximum clock frequency of the design, a behavioral model of the circuit in Figure 4.3 was written in Verilog HDL. A model for each of the major functional blocks in the circuit was written and simulated individually using the Verilog simulator. After each block was working correctly, the entire circuit was simulated at the top level. To verify correct operation, data was read in to the simulator from the same image files used to do the C code simulations. The results obtained from the C code were compared with the hardware simulation results to verify that they matched perfectly.

After the Verilog model was verified, it was synthesized into gates using the Synopsys synthesis compiler. The design was implemented in a 0.5 μm CMOS process using a standard cell methodology. This is the same process and methodolgy used for the existing flowgraph design. Table 5.2 shows an estimate of the gate count for each major functional block in the circuit. The total gate count of 9,559 gates compares favorably to the count of 11.2k gates for the existing flowgraph design.

Table 5.2: Gate Count for the DA 2-D DCT Circuit

Input Multiplexers	63
Input Registers	1178
PISO Shift Registers	1744
Bit-Serial Butterfly	166
Lookup Tables	448
Accumulators	2883
Output Registers	1575
IDCT Butterfly	753
8-Bit Limiter	47
Control	702
Total	9559

An interesting point about this architecture is that the flip-flops in the input registers, PISO shift registers, accumulators and output registers use 56% of the area for the design. The main reason that so many flip-flops are required is that

there are 2 sets of registers in the input and the output of the matrix multiplier. These are necessary because the input samples must all be available before the matrix multiplication can begin and the output results are all presented on the same cycle.

One way to reduce the gate count would be to replace flip-flops with latches. In particular, the PISO shift registers and the output registers are clocked only once every 16 cycles. Implementing these elements with latches would reduce the gate count by approximately 1280 gates. The disadvantage to using latches is that the complexity of a scan-based test program would be increased. Another possible way to reduce the gate count would be to restructure the architecture so that the outputs come out one at a time instead of all at once. This is an area for further research.

While the DA architecture appears to use less die area than the flowgraph design, one caveat should be noted. Flowgraph architectures may have an advantage in video and image compression applications where quantization follows the DCT. It is possible to merge the quantization coefficients into the coefficients of the flowgraph algorithm so that a separate quantization circuit is not required. Linzer and Feig [17] describe this technique. For a DA architecture, it is generally not practical to merge the quantization coefficients into the constants stored in the lookup tables. Doing so increases the lookup memory required by a factor of N. This increase would make the DA implementation larger than the flowgraph implementation. To compare the area of the DA architecture and the flowgraph architecture for these applications, the DCT and quantization circuit must be considered together.

Clock Rate

The maximum clock rate for this design is 32.6 MHz. This estimate is based on gate delays of the synthesized circuit and pessimistic wire delay estimates. This clock rate almost matches the clock rate of 33 MHz for the existing flowgraph design. It is estimated that the clock rate of the DA design would be equal to or greater than 33 MHz using wire delays back-annotated from the layout. The critical path in

the design is from the output of the PISO shift registers to the input of the registers in the accumulators. The two major delays in this path are the lookup memory (6.1 ns) and the adders in the accumulators (23.3 ns). Since the synthesis process was constrained for minimum area, ripple carry adders were used in the accumulators. No attempt was made to optimize the layout for speed.

Another advantage of this DA architecture is that it offers greater flexibility to design for a shorter clock cycle time. The flowgraph architecture requires a single-cycle multiplier which cannot be pipelined. As the clock cycle time decreases, it becomes increasingly difficult to design a single-cycle multiplier. The DA architecture could easily be pipelined by inserting a register between the lookup tables and the accumulators. To further increase speed, the adders in the accumulators could be changed to carry-lookahead adders. It is estimated that these changes would add about 1600 gates to the design and increase speed to at least 66 MHz.

Conclusions

A DCT/IDCT circuit based on distributed arithmetic has been presented which has novel improvements in the lookup tables. The architecture has been simulated using Matlab and C code models to verify that it has sufficient accuracy for use in image compression applications. A VLSI implementation was done which achieves the same clock speed and throughput as a comparable circuit based on a flowgraph algorithm and uses 14.6% less area.

BIBLIOGRAPHY

[1] J.C. Carlach, et. al., "TCAD: A 27 MHz 8x8 Discrete Cosine Transform Chip", IEEE Proc. ICASSP, Vol. 4, pp. 2429-2432, 1989.

[2] T.C. Chen, et. al., "VLSI Implementation of a 16 x 16 DCT", IEEE Proc. ICASSP, pp. 1973-1976, 1988.

[3] W.H. Chen, C. H. Smith and S. C. Fralick, "A Fast Computational Algorithm for the Discrete Cosine Transform", IEEE Transactions on Communications, vol COM-25, No. 9, pp. 1004-1009, September 1977.

[4] N.I. Cho and S.U. Lee, "Fast Algorithm and Implementation of 2-D Discrete Cosine Transform", IEEE Transactions on Circuits and Systems, Vol. 38, No. 3, pp. 297-305, March 1991.

[5] S. Cismas, "System and Method for IDCT", United States Patent 5,574,661. Filed July 29, 1994, granted November 12, 1996.

[6] N. Demassieux, et. al., "An Optimized VLSI Architecture for A Multiformat Discrete Cosine Transform", IEEE Proc. ICASSP, Vol. 1, pp. 547-550, 1987.

[7] P. Duhamel and C. Guillemot, "Polynomial Transform Computation of 2-D DCT", IEEE Proc. ICASSP, Vol. 3, pp. 1515-1518, 1990.

[8] E. Feig, "A Fast Scaled DCT Algorithm", Proc. SPIE-SPSE, Vol. 1244, Image Processing Algorithms and Techniques, Santa Clara, CA, February 1990.

[9] A. Jain, Fundamentals of Digital Image Processing, Prentice Hall, Englewood Cliffs, N.J., 1989.

[10] P.C. Jain, W. Schlenk and M. Riegel, "VLSI Implementation of Two-Dimensional DCT Processor in Real-Time for Video Codec", IEEE Trans. Consumer Electronics, vol. 38, pp. 537-544, August 1992.

[11] M.A. Haque, "A Two-Dimensional Fast Cosine Transform", IEEE Trans. Acoust., Speech, Signal Process., Vol. ASSP-33, pp. 1532-1539, 1985.

[12] H.S. Hou, "A Fast Recursive Algorithm for Computing the Discrete Cosine Transform", IEEE Trans. Acoust., Speech, Signal Process., Vol. ASSP-35, pp. 1455-1461, 1987.

[13] F.A. Kamangar and K.R. Rao, "Fast Algorithms for the 2-D Discrete Cosine Transform", IEEE Trans. Comput., Vol. C-31, pp. 899-906, September 1982.

[14] B.G. Lee, "A New Algorithm to Compute the Discrete Cosine Transform", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol ASSP-32, No. 6, December 1984.

[15] Q. Li, "Design and Performance Estimation of 2-D Discrete Cosine Transform", M.S. Thesis, Electrical and Computer Engineering Department, Oregon State University, Spring 1996.

[16] W. Li, "A New Algorithm to Compute the DCT and its Inverse", IEEE Transactions on Signal Processing, Vol. 39, No. 6, June 1991.

[17] E. Linzer and E. Feig, "New Scaled DCT Algorithms for Fused Multiply/Add Architectures", IEEE Proc. ICASSP, Vol. 3, pp 2201-2204, 1991.

[18] C. Loeffler, et.al, "Practical Fast 1-D DCT Algorithms with 11 Multiplications", IEEE Proc. ICASSP, Vol. 2, pp. 988-991, 1989.

[19] A. Madisetti and A. Willson, Jr., "A 100 MHz 2-D 8 x 8 DCT/IDCT Processor for HDTV Applications", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 5, No. 2, pp 158-165, April 1995.

[20] J. Makhoul, "A Fast Cosine Transform in One and Two Dimensions", IEEE Trans. Acoust., Speech and Signal Proces., Vol. ASSP-32, pp. 27-34, February 1980.

[21] T. Masaki, et. al., "VLSI Implementation of Inverse Discrete Cosine Transformer and Motion Compensator for MPEG2 HDTV Video Decoding", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 5, No. 5, pp 387-395, October 1995.

[22] A. Peled and B. Liu, "A New Hardware Realization of Digital Filters", IEEE Trans. Acoust., Speech and Signal Proces., Vol ASSP-22, No. 6, pp. 456-462, December 1974.

[23] J.L. Mitchell, et.al., MPEG Video Compression Standard, Chapman and Hall, New York, NY, 1997.

[24] P. Pirsch, et. al., "VLSI Architectures for Video Compression - A Survey", Proceedings of the IEEE, Vol. 83, No. 2, pp. 220-246, February 1995.

[25] K.R. Rao and J.J. Hwang, Techniques and Standards for Image, Video & Audio Coding, Prentice Hall, Upper Saddle River, NJ, 1996.

[26] K.R. Rao and P. Yip, Discrete Cosine Transform: Algorithms, Advantages, Applications Academic Press, Boston, MA, 1990.

[27] M. Sheu, et. al., "A High Throughput-Rate Architecture for 8 x 8 2-D DCT", IEEE International Symposium on Circuits and Systems, pp. 1587-1590, 1993.

[28] D. Slawecki and W. Li, "DCT/IDCT Processor Design for High Data Rate Image Coding", IEEE Transactions on Circuits and Systems, Vol. 2, No. 2, pp. 135-144, June 1992.

[29] V. Srinivasan and K.J.R. Liu, "VLSI Design of High-Speed Time-Recursive 2-D DCT/IDCT Processor for Video Applications", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 6, No. 1, pp 87-95, February 1996.

[30] M.T. Sun, et. al., "A Concurrent Architecture for VLSI Implementation of Discrete Cosine Transform", IEEE Transactions on Circuits and Systems, Vol. CAS-34, No. 8, August 1987.

[31] Uramoto, S., et. al., "A 100 MHz 2-D Discrete Cosine Transform Core Processor", IEEE Journal of Solid-State Circuits, Vol. 27, No. 4, pp. 492-498, April 1992.

[32] M. Vetterli and H. Nussbaumer, "Simple FFT and DCT Algorithms with Reduced Number of Operations", Signal Processing, Vol. 6, No. 4, pp. 267-278, August 1984.

[33] G.K. Wallace, "The JPEG Still Image Compression Standard", Communications of the ACM, Vol. 34, No. 4, pp. 31-44, April 1991.

[34] C. Wang and C. Chen, "High-Throughput VLSI Architectures for the 1-D and 2-D Discrete Cosine Transforms", IEEE Transactions on Circuits and Systems, Vol. 5, No. 1, pp. 31-40, February 1995.

[35] S.A. White, "Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review", IEEE ASSP Magazine, Vol. 6, No. 3, pp. 4-18, July 1989.