DESIGN OF 2D DISCRETE COSINE TRANSFORM USING CORDIC ARCHITECTURES IN VHDL

A THESIS SUBMITTED IN PARTIAL FULFILMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
In
VLSI design and embedded system

By
J.SriKrishna
Roll No: 20507004

Department of Electronics and Communication Engineering
National Institute of Technology, Rourkela
May, 2007

DESIGN OF 2D DISCRETE COSINE TRANSFORM USING CORDIC ARCHITECTURES IN VHDL

A THESIS SUBMITTED IN PARTIAL FULFILMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
In
VLSI design and embedded system

By
J. Srikrishna
Roll No: 20507004

Under the guidance of
Prof. K.K. Mahapatra

Department of Electronics and Communication Engineering
National Institute of Technology, Rourkela
May, 2007

National Institute of Technology
Rourkela

CERTIFICATE

This is to certify that the thesis titled, "Design of 2D Discrete Cosine Transform using CORDIC architectures in VHDL" submitted by J.SriKrishna in partial fulfillment of the requirements for the award of Master of Technology Degree in Electronics and communication Engineering with specialization in "VLSI design and Embedded system" at the National Institute of Technology, Rourkela (Deemed University) is an authentic work carried out by him under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other university / institute for the award of any Degree or Diploma.

Date:

Prof. K. K. Mahapatra
Dept. of Electronics & Comm. Engineering
National Institute of Technology, Rourkela
Pin – 769008

National Institute of Technology

Rourkela

ACKNOWLEDGEMENTS

I am thankful to **Dr. K. K. Mahapatra,** Professor in the department of Electronics and Communication Engineering, NIT Rourkela for giving me the opportunity to work under him and lending every support at every stage of this project work.

I would also like to convey my sincerest gratitude and indebt ness to all other faculty members and staff of Department of Electronics and Communications Engineering, NIT Rourkela, who bestowed their great effort and guidance at appropriate times without which it would have been very difficult on my part to finish the project work. I also very thankful to all my class mates and friends of VLSI lab-I especially sushant (M.Tech(R)), Jitendra Das (Phd scholar), Durga who always encouraged me in the successful completion of my thesis work.

Finally, I wish to express my eternal indebtedness to my parents and my brother for the unflagging encouragement and support that I have received through the years and to whom I owe a lot.

Date

J.SriKrishna

Dept. of Electronics & Communications Engineering

National Institute of Technology, Rourkela

Pin - 769008

CONTENTS

Abstract

The Discrete Cosine Transform is one of the most widely transform techniques in digital signal processing. In addition, this is also most computationally intensive transforms which require many multiplications and additions. Real time data processing necessitates the use of special purpose hardware which involves hardware efficiency as well as high throughput. Many DCT algorithms were proposed in order to achieve high speed DCT. Those architectures which involves multipliers ,for example Chen's algorithm has less regular architecture due to complex routing and requires large silicon area. On the other hand, the DCT architecture based on distributed arithmetic (DA) which is also a multiplier less architecture has the inherent disadvantage of less throughputs because of the ROM access time and the need of accumulator. Also this DA algorithm requires large silicon area if it requires large ROM size. Systolic array architecture for the real-time DCT computation may have the large number of gates and clock skew problem. The other ways of implementation of DCT which involves in multiplierless, thus power efficient and which results in regular architecture and less complicated routing, consequently less area, simultaneously lead to high throughput. So for that purpose CORDIC seems to be a best solution. CORDIC offers a unified iterative formulation to efficiently evaluate the rotation operation.

This thesis presents the implementation of 2D Discrete Cosine Transform (DCT) using the Angle Recoded (AR) Cordic algorithm, the new scaling less CORDIC algorithm and the conventional Chen's algorithm which is multiplier dependant algorithm. The 2D DCT is implemented by exploiting the Separability property of 2D Discrete Cosine Transform. Here first one dimensional DCT is designed first and later a transpose buffer which consists of 64 memory elements, fully pipelined is designed. Later all these blocks are joined with the help of a controller element which is a mealy type FSM which produces some status signals also. The three resulting architectures are all well synthesized in Xilinx 9.1ise, simulated in Modelsim 5.6f and the power is calculated in Xilinx Xpower. Results prove that AR Cordic algorithm is better than Chen's algorithm, even the new scaling less CORDIC algorithm.

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Digital signal processing (DSP) algorithms exhibit an increasing need for the efficient implementation of complex arithmetic operations. The computation of trigonometric functions, coordinate transformations or rotations of complex valued phasors is almost naturally involved with modern DSP algorithms. In this thesis one of the most computationally high algorithm called the Discrete Cosine Transform is implemented with the help CORDIC(Co ordinate Rotation Digital computer) algorithm which results in a multiplier less architectures and comparison is made between the DCT using Chen's algorithm and DCT using CORDIC as well as new CORDIC algorithm.

1.1 Motivation

Discrete cosine transform (DCT) is widely used transform in image processing, especially for compression. Some of the applications of two-dimensional DCT involve still image compression and compression of individual video frames, while multidimensional DCT is mostly used for compression of video streams and volume spaces. Transform is also useful for transferring multidimensional data to DCT frequency domain, where different operations, like spread-spectrum data watermarking, can be performed in easier and more efficient manner. A countless number of papers discussing DCT algorithms is strongly witnessing about its importance and applicability.

Hardware implementations are especially interesting for the realization of highly parallel algorithms that can achieve much higher throughput than software solutions. In addition, special purpose DCT hardware discharges the computational load from the processor and therefore improves the performance of complete multimedia system. The throughput is directly influencing the quality of experience of multimedia content. Another important factor that influences the quality of is the finite register length effect on the accuracy of the forward-inverse transformation process.

The Discrete Cosine Transform is one of the most widely transform techniques in digital signal processing. Hence the motivation for the design of the Discrete Cosine transform architecture is clear. As this is also most computationally intensive transforms which require many multiplications and additions. Real time data processing necessitates the use of special purpose hardware which involves hardware efficiency as well as high throughput. Many DCT algorithms were proposed in order to achieve high speed DCT. Those architectures which involves multipliers ,for example Chen's algorithm has less regular architecture due to complex routing and requires large silicon area. On the other hand, the DCT architecture based on distributed arithmetic (DA) which is also a multiplier less architecture has the inherent disadvantage of less throughputs because of the ROM access

time and the need of accumulator. Also this DA algorithm requires large silicon area if it requires large ROM size. Systolic array architecture for the real-time DCT computation may have the large number of gates and clock skew problem. The other ways of implementation of DCT which involves in multiplierless, thus power efficient and which results in regular architecture and less complicated routing, consequently less area, simultaneously lead to high throughput. So for that purpose CORDIC seems to be a best solution. CORDIC offers a unified iterative formulation to efficiently evaluate the rotation operation.

In CORDIC suppose if the desired operation is to multiply two complex numbers say $X+jY$ and $\cos(\theta)+j\sin(\theta)$, so that the resultant operation is $x\cos(\theta)-y\sin(\theta)$ and another o/p is $x\sin(\theta)+y\cos(\theta)$, we can perform very effectively without using much overhead ,i.e. without multipliers and only with the help of simple shift and add operations and small scaling. The details of the algorithm are given in chapter[4].In each iteration we have to perform some rotations and additions. More iterations much more accuracy and therefore less error. In this algorithm, each angle is approximated in terms of small angles $\tan^{-1}(2^{-i})$ where 'i' denotes each iteration and these set of angles are placed in a LUT. But the conventional CORDIC has some inherent disadvantages of more number of iterations and very hard scaling factor. So another algorithm by name AR CORDIC algorithm is implemented which contains less number of iterations and less complex scaling factor, since the scaling is also implemented with the help of shift and add operations only. So it's a better solution. But another algorithm is there by name THE NEW CORDIC algorithm which doesn't have no scaling factor, but to have particularly good precision, we need to have more bitwidth up to 50 bits which is considerably more number, which occupies more Input Output Buffers (IOBs), consequently more area and more power.

1.2Literature Review

In this thesis, the principle reference is reference [4] titled "Low-power Multiplierless DCT architecture using Image Data Correlation". The AR CORDIC algorithm is brought from that reference and of course later modified. Those modifications are in reducing in the number of iterations and scaling factor. Later the main idea about CORDIC is from references [3], [5], [6], [7], [8].Those references are correctly described about CORDIC. Later the VLSI implementations and architectures are in reference [9], [17], [19], where Yu Hen Hu correctly described about the different architectures of CORDIC in VLSI and FU also described the architectures. Later the different implementations of DCT in VLSI are given in reference [11] which is a tutorial like this. Before that all the references about DCT

is given in reference [16], [21], where they give a brief understand of DCT. Later the design of another important block in the design of DCT Core i.e. Transpose buffer is given in reference [12].The VHDL tutorial is given from reference [1], [2] which gives a good understanding of VHDL.

1.3Overview of Thesis

The next chapter discusses about the DCT-An overview and chapter [3] discusses about the different implementations of DCT. Chapter [4] discusses about in detail of CORDIC algorithm. Chapter [5] discusses about the different architectures of CORDIC. Later chapters discusses about the fundamentals of Low Power Design. Chapter [7] describes about the design of Discrete Cosine transform (DCT) in details about the I/O bitwidth and the architecture and block diagram of it is also mentioned. Chapter [8] describes the simulation results.

Chapter 2

DISCRETE COSINE TRANSFORM
AN OVERVIEW

The Discrete Cosine Transform (DCT) was first proposed by Ahmed et al. (1974), and it has been more and more important in recent years. DCT has been widely used in signal processing of image data, especially in coding for compression, especially in lossy compression, for its near-optimal performance. Because of the wide-spread use of DCT's, research into fast algorithms for their implementation has been rather active ,and also, since the DCT is computation intensive, the development of highspeed hardware and real-time DCT processor design have been object of research .

Discrete cosine transform (DCT) is widely used in image processing, especially for compression. Some of the applications of two-dimensional DCT involve still image compression and compression of individual video frames, while multidimensional DCT is mostly used for compression of video streams. DCT is also useful for transferring multidimensional data to frequency domain, where different operations, like spread-spectrum, data compression, data watermarking, can be performed in easier and more efficient manner. A number of papers discussing DCT algorithms is available in the literature that signifies its importance and application.

Hardware implementation of parallel DCT transform is possible, that would give higher throughput than software solutions. Special purpose DCT hardware decreases the computational load from the processor and therefore improves the performance of complete multimedia system. The throughput is directly influencing the quality of experience of multimedia content. Another important factor that influences the quality is the finite register length effect that affects the accuracy of the forward-inverse transformation process.

Hence, the motivation for investigating hardware specific DCT algorithms is clear. As 2-D DCT algorithms are the most typical for image compression, the main focus of this chapter will be on the efficient hardware implementations of 2-D DCT based compression by decreasing the number of computations, increasing the accuracy of reconstruction, and reducing the chip area. This in return reduces the power consumption of the compression technique. As the number of applications that require higher-dimensional DCT algorithms are growing, a special attention will be paid to the algorithms that are easily extensible to higher dimensional cases. The JPEG standard has been around since the late 1980's and has been an effective first solution to the standardisation of image compression. Although JPEG has some very useful strategies for DCT quantisation and compression, it was only developed for low

compressions. The 8×8 DCT block size was chosen for speed (which is less of an issue now, with the advent of faster processors) not for performance.

Like other transforms, the Discrete Cosine Transform (DCT) attempts to decorrelate the image data. After decorrelation each transform coefficient can be encoded independently without losing compression efficiency. This section describes the DCT and some of its important properties.

2.1The One-Dimensional DCT

The most common DCT definition of a 1-D sequence of length N is

$$C(u) = \alpha(u)\sum_{x=0}^{N-1} f(x)\cos\left[\frac{\pi(2x+1)u}{2N}\right] \qquad (1)$$

for u = 0,1,2,…., N — 1. Similarly, the inverse transformation is defined as

$$f(x) = \sum_{u=0}^{N-1} \alpha(u)c(u)\cos\left[\frac{\pi(2x+1)u}{2N}\right] \qquad (2)$$

for x = 0,1,2..,, N — 1. In both equations (1) and (2) $\alpha(u)$ is defined as

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & for \quad u = 0 \\ \sqrt{\frac{2}{N}} & for \quad u \neq 0. \end{cases} \qquad (3)$$

It is clear from (1) that for u = 0, $c(u=0) = \sqrt{\frac{1}{N}}\sum_{x=0}^{N-1} f(x)$. Thus, the first transform coefficient is the average value of the sample sequence. In literature, this value is referred to as the *DC Coefficient.* All other transform coefficients are called the *AC Coefficients.*

To fix ideas, ignore the $f(x)$ and $\alpha(u)$ component in (1). The plot of $\sum_{x=0}^{N-1}\cos\left[\frac{\pi(2x+1)u}{2N}\right]$ for N = 8 and varying values of u is shown in Figure 1. In accordance with our previous observation, the first the top-left waveform (u = 0) renders a constant (DC) value, whereas, all other waveforms (u = 1,2,...,) give waveforms at progressively increasing frequencies . These waveforms are called the *cosine basis function.* Note that these basis functions are orthogonal. Hence, multiplication of any waveform in Figure 3 with another waveform followed by a summation over all sample points yields a zero (scalar) value, whereas

multiplication of any waveform in Figure 1 with itself followed by a summation yields a constant (scalar) value. Orthogonal waveforms are independent, that is, none of the basis functions can be represented as a combination of other basis functions .

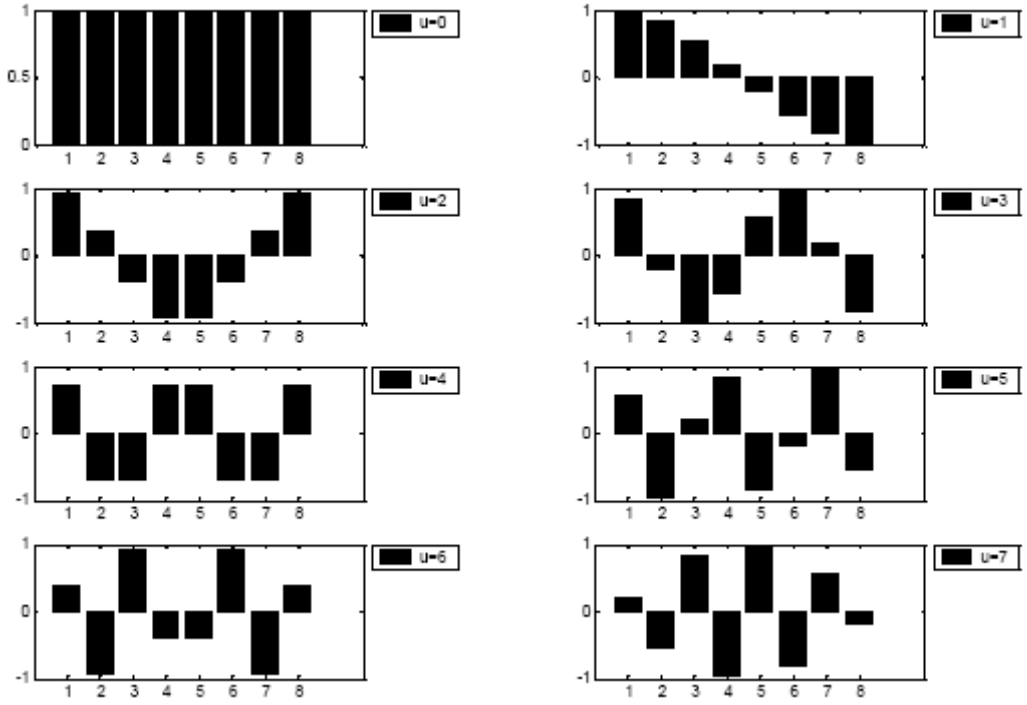


Fig 2.1.One Dimensional Cosine Functions

If the input sequence has more than *N* sample points then it can be divided into sub-sequences of length *N* and DCT can be applied to these chunks independently. Here, a very important point to note is that in each such computation the values of the basis function points will not change. Only the values of fx() will change in each sub-sequence. This is a very important property, since it shows that the basis functions can be pre-computed offline and then multiplied with the sub-sequences. This reduces the number of mathematical operations (i.e., multiplications and additions) thereby rendering computation efficiency.

2.2The Two-Dimensional DCT

The objective of this document is to study the efficacy of DCT on images. This necessitates the extension of ideas presented in the last section to a two-dimensional space. The 2-D DCT is a direct extension of the 1-D case and is given by

$$C(u,v) = \frac{1}{4}\alpha(u)\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\frac{\cos(2x+1)u\pi}{2N}\frac{\cos(2y+1)v\pi}{2N} \qquad (4)$$

for $u,v = 0,1,2,\ldots,N-1$ and $\alpha(u)$ and $\alpha(v)$ are defined in (3). The inverse transform is

defined as

$$f(x,y)=\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}\alpha(u)\alpha(v)C(u,v)\cos\left[\frac{\pi(2x+1)u}{2N}\right]\cos\left[\frac{\pi(2y+1)v}{2N}\right] \quad (5)$$

for $x,y = 0,1,2,\ldots N-1$. The 2-D basis functions can be generated by multiplying the horizontally oriented 1-D basis functions (shown in Figure 1) with vertically oriented set of the same functions [13]. The basis functions for $N = 8$ are shown in. Again, it can be noted that the basis functions exhibit a progressive increase in frequency both in the vertical and horizontal direction. The top left basis function of results from multiplication of the DC component in Figure 1 with its transpose. Hence, this function assumes a constant value and is referred to as the DC coefficient.

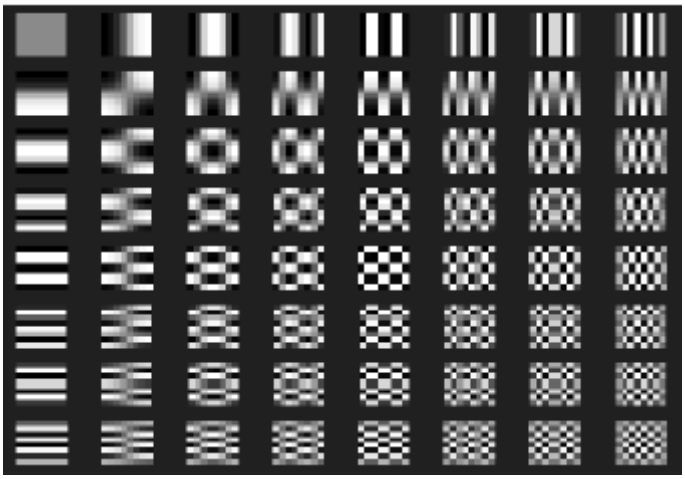


Fig 2.2. Two dimensional DCT basis functions (N = 8). Neutral gray represents zero, white represents positive amplitudes, and black represents negative amplitude

2.3Properties of DCT

Discussions in the preceding sections have developed a mathematical foundation for DCT. However, the intuitive insight into its image processing application has not been presented. This section outlines (with examples) some properties of the DCT which are of particular value to image processing applications.

2.3.1Decorrelation

As discussed previously, the principle advantage of image transformation is the removal of redundancy between neighboring pixels. This leads to uncorrelated transform coefficients which can be encoded independently. The normalized autocorrelation of the images before and after DCT is shown in Figure 3. Clearly, the amplitude of the autocorrelation after the

DCT operation is very small at all lags. Hence, it can be inferred that DCT exhibits excellent decorrelation properties.

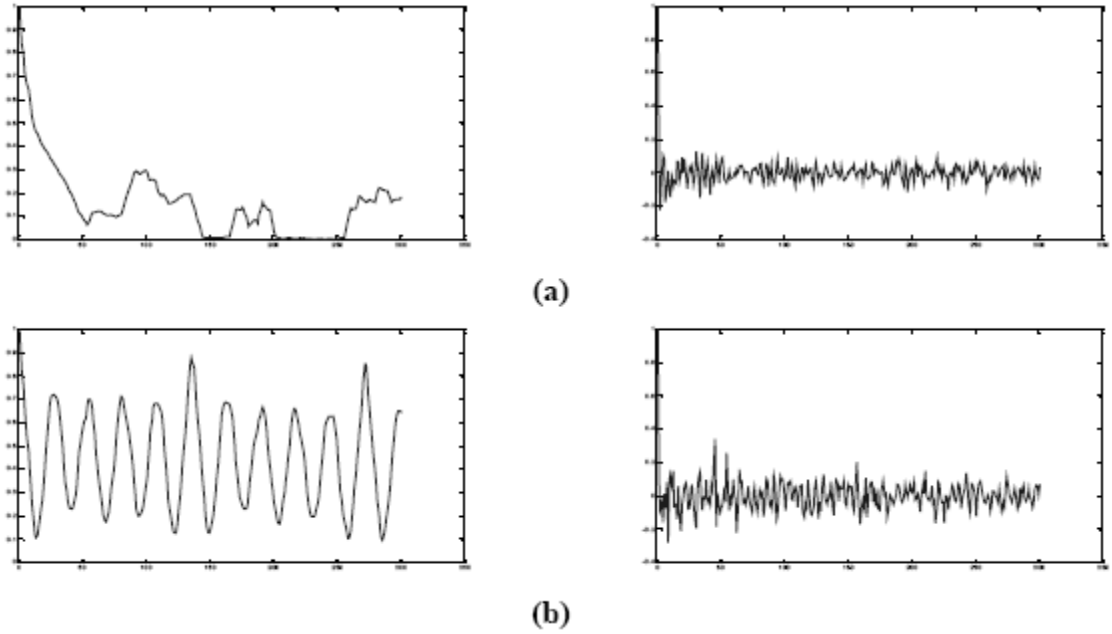


Fig 2.3. (a) Normalized autocorrelation of uncorrelated image before and after DCT; (b) Normalized autocorrelation of correlated image before and after DCT.

2.3.2Energy Compaction

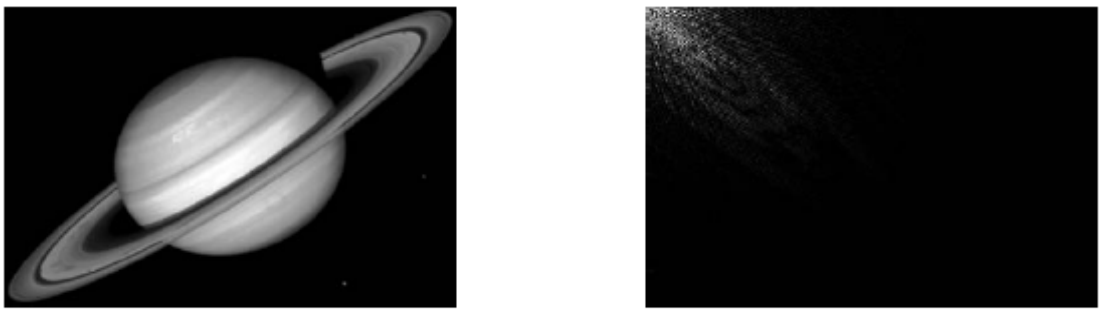
Efficacy of a transformation scheme can be directly gauged by its ability to pack input data into as few coefficients as possible. This allows the quantizer to discard coefficients with relatively small amplitudes without introducing visual distortion in the reconstructed image. DCT exhibits excellent energy compaction for highly correlatedimages

.

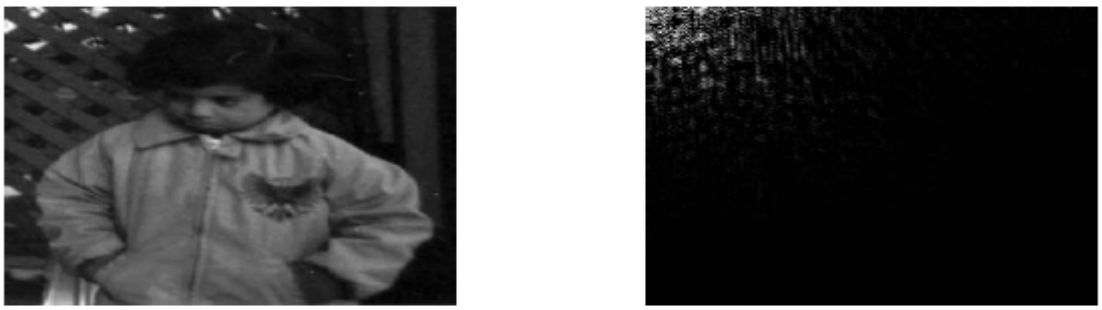
Let us again consider the two example images (a) and (b). In addition to their respective correlation properties discussed in preceding sections, the uncorrelated image has more sharp intensity variations than the correlated image. Therefore, the former has more

high frequency content than the latter. Figure 6 shows the DCT of both the images. Clearly, the uncorrelated image has its energy spread out, whereas the energy of the correlated image is packed into the low frequency region (i.e., top left region).

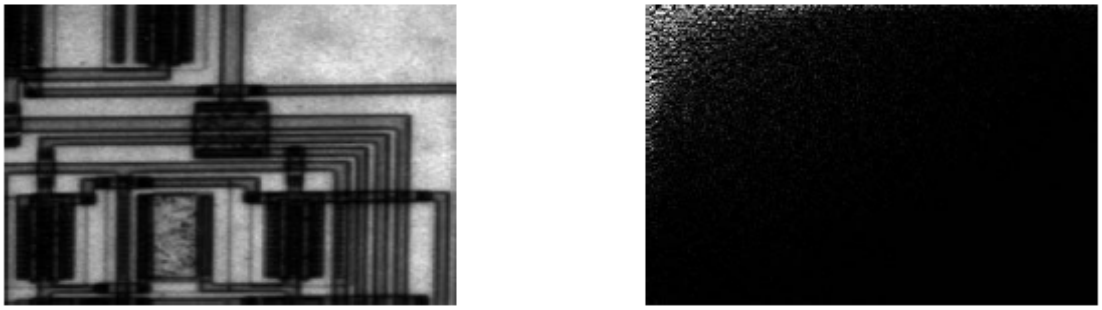
Other examples of the energy compaction property of DCT with respect to some standard images are provided in Figure



(a)



(b)



(c)

4.

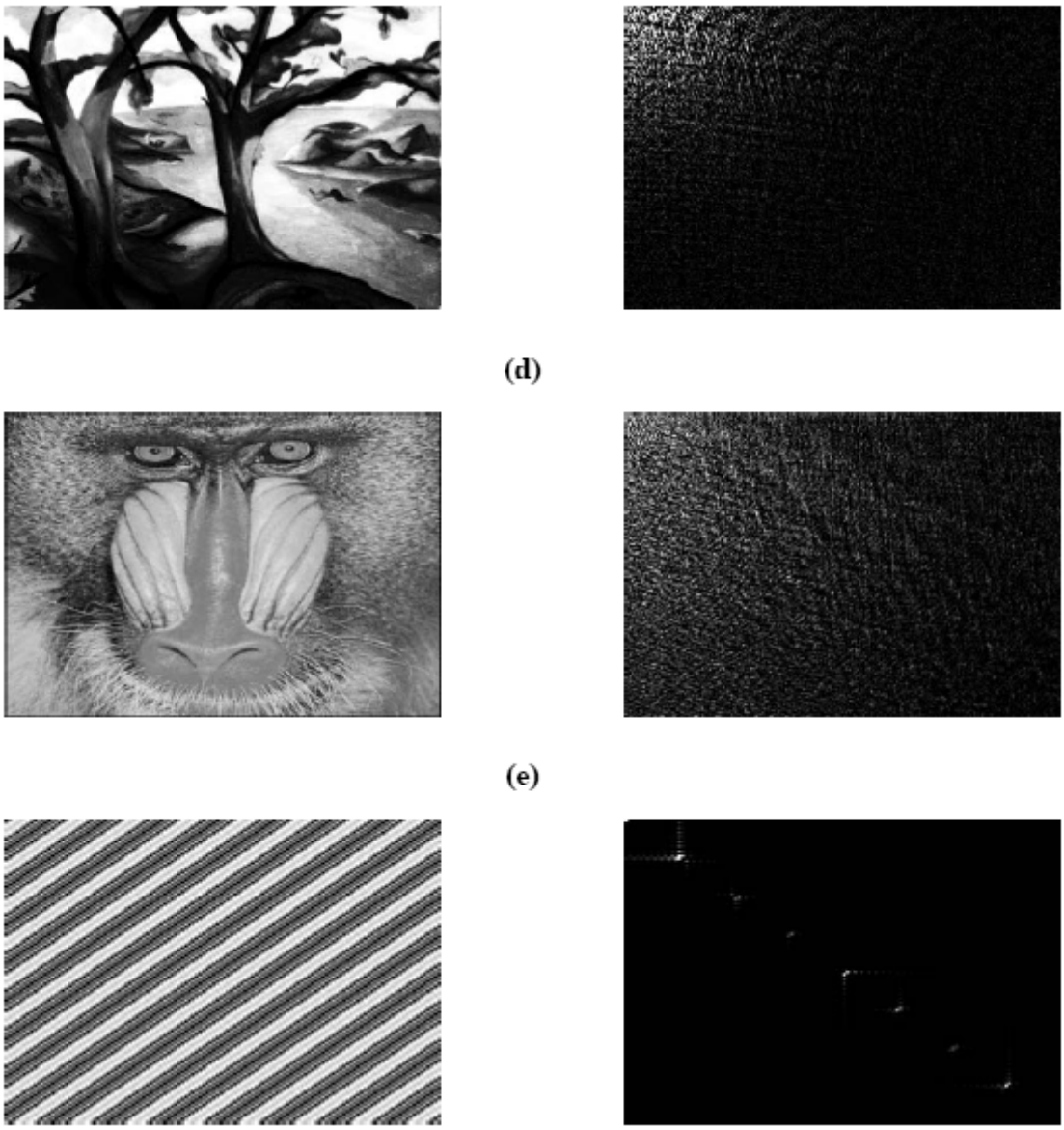


Fig 2.4. (a) Saturn and its DCT; (b) Child and its DCT; (c) Circuit and its DCT; (d) Trees and its DCT; (e) Baboon and its DCT; (f) a sine wave and its DCT.

A closer look at Figure 4 reveals that it comprises of four broad image classes. Figure 2.4(a) and 2.4(b) contain large areas of slowly varying intensities. These images can be classified as low frequency images with low spatial details. A DCT operation on these images provides very good energy compaction in the low frequency region of the transformed image. Figure 4(c) contains a number of edges (i.e., sharp intensity variations) and therefore can be classified as a high frequency image with low spatial content. However, the image data exhibits high correlation which is exploited by the DCT algorithm to provide good energy compaction. Figure 4 (d) and (e) are images with progressively high frequency and spatial content. Consequently, the transform coefficients are spread over low and high frequencies.

Figure 4(e) shows periodicity therefore the DCT contains impulses with amplitudes proportional to the weight of a particular frequency in the original waveform. The other (relatively insignificant) harmonics of the sine wave can also be observed by closer examination of its DCT image.

Hence, from the preceding discussion it can be inferred that DCT renders excellent energy compaction for correlated images. Studies have shown that the energy compaction performance of DCT approaches optimality as image correlation approaches one i.e., DCT provides (almost) optimal decorrelation for such images.

2.3.3 Separability

The DCT transform equation (4) can be expressed as,

$$C(u,v) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\left[\frac{\pi(2x+1)u}{2N}\right]\cos\left[\frac{\pi(2y+1)v}{2N}\right] \qquad (6)$$

for u,v = 0,1,2,...,N-1.

This property, known as *separability,* has the principle advantage that *C (u,* v) can be computed in two steps by successive 1-D operations on rows and columns of an image. This idea is graphically illustrated in Figure 5. The arguments presented can be identically applied for the inverse DCT computation (5).For the hardware design, this property is utilized in this project.

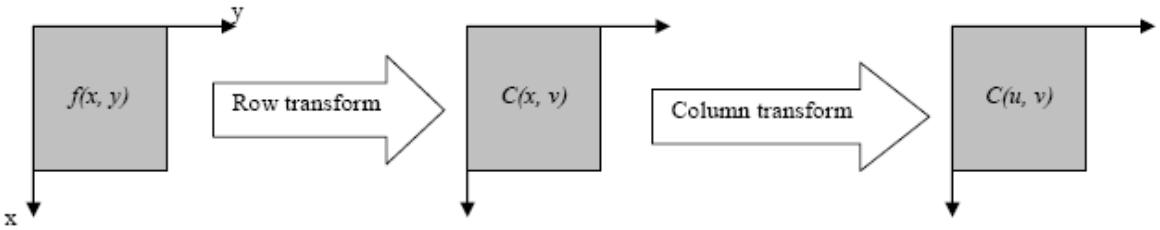


Fig 2.5. Computation of 2-D DCT using separability property.

2.3.4 Symmetry

Another look at the row and column operations in Equation 6 reveals that these operations are functionally identical. Such a transformation is called a *symmetric transformation.* A separable and symmetric transform can be expressed in the form .

$$T = AfA \qquad (7)$$

Where A is an N XN symmetric transformation matrix with entries

$$\alpha(i,j) = \alpha(j)\sum_{j=0}^{N-1}\cos\left[\frac{\pi(2j+1)i}{2N}\right]$$ and f is the NXN image matrix.

This is an extremely useful property since it implies that the transformation matrix can be pre-computed offline and then applied to the image thereby providing orders of magnitude improvement in computation efficiency.

2.3.5 Orthogonality

In order to extend ideas presented in the preceding section, let us denote the inverse transformation of (7) as $f = A^{-1}TA^{-1}$.

As discussed previously, DCT basis functions are orthogonal (See Section 2.1). Thus, the inverse transformation matrix of A is equal to its transpose i.e. $A^{-1} = A^{T}$. Therefore, and in addition to its decorrelation characteristics, this property renders some reduction in the pre-computation complexity.

Chapter 3

Different implementations of Discrete Cosine Transform

Implementation of the 2-D DCT directly from the theoretical equation (equation 3.2), results in 1024 multiplications and 896 additions. Fast algorithms exploit the symmetry within the DCT to achieve dramatic computational savings.

There are three basic categories of approach for computation of the 2-D DCT. The first category of 2-D DCT implementation is indirect computation through other transforms—most commonly, the Discrete Hartley Transform (DHT) and the Discrete Fourier Transform (DFT). The DHT-based algorithm of shows increased performance in throughput, latency, and turnaround time. Optimization with respect to these parameters is not the focus of the proposed project. A DFT approach [5] calculates the odd-length DCT, which is not applicable to this project since the design must be compatible with JPEG standards.

The second style of algorithms computes the 2-D DCT by row-column decomposition. In this approach, the separability property of the DCT is exploited. An 8-point, 1-D DCT is applied to each of the 8 rows, and then again to each of the 8 columns. The 1-D algorithm that is applied to both the rows and columns is the same. Therefore, it could be possible to use identical pieces of hardware to do the row computation as well as the column computation. A transposition matrix would separate the two as the functional description in figure 3.1 shows. The bulk of the design and computation is in the 8 point 1-D DCT block, which can potentially be reused 16 times—8 times for each row, and 8 times for each column. Therefore, the fast algorithm for computing the 1-D DCT is usually selected. The high regularity of this approach is very attractive for reduced cell count and easy very large scale integration (VLSI) implementation.

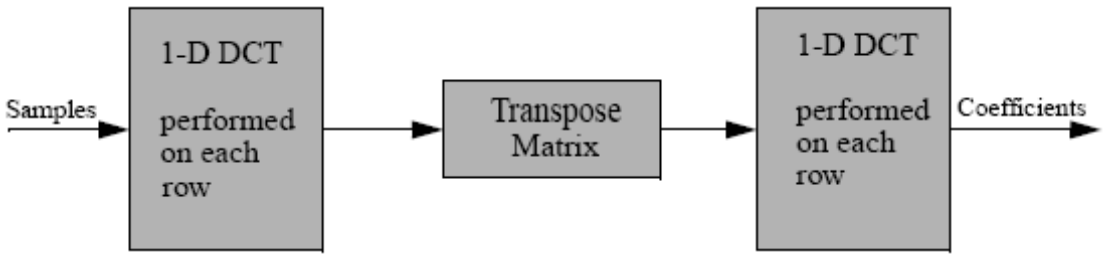


Figure 3.1:2-D DCT implementation

The third approach to computation of the 2-D DCT is by a direct method using the results of a polynomial transform. Computational complexity is greatly reduced, but regularity is sacrificed. Instead of the 16 1-D DCTs used in the conventional row-column decomposition, [6] uses all real arithmetic including 8 1-D DCTs, and stages of pre-adds and

post-adds (a total of 234 additions) to compute the 2-D DCT. Thus, the number of multiplications for most implementations should be halved as multiplication only appears within the 1-D DCT. Although this direct method of extension into two dimensions creates an irregular relationship between inputs and outputs of the system, the savings in computational power may be significant with the use of certain 1-D DCT algorithms. With this direct approach, large chunks of the design cannot be reused to the same extent as in the conventional row-column decomposition approach. Thus, the direct approach will lead to more hardware, more complex control, and much more intensive debugging.

Since row-column decomposition is very useful for VLSI implementation, that implementation is considered in this project. In that implementation, starts with one-dimensional transform. So at first the implementation of fast 1-D transforms are considered .There are three algorithms considered in this project. They are well tabulated in the table below.

1-D algorithm	Multiplications	Additions	Compensation
Theoretical Equations(2.1)	64	56	Not applicable
Chen and Fralick	32	32	Not Applicable
Using CORDIC algorithm	0	12	Depends on angle
Using New CORDIC algorithm by Zhang	0	12	No compensation

Table 3.1:Comparison of three algorithms in terms of Multiplications and additions

Certain 1-D DCT algorithms become more optimal in the row-column approach when it is known that DCT calculation will be followed by quantization. In these cases, the numbers of multiplications are reduced by incorporating multiplications within the final stage of a 1-D DCT algorithm into the quantization table. When the Agui, Arai, and Nakajima 1-D DCT described ,is implemented in the row-column fashion, as few as 144 multiplications and 464 additions are needed .For this particular algorithm, a savings of 8 multiplications per 1-D DCT calculation on each column can be saved, for a total savings of 64 multiplications on the

2-D DCT computation. The reduction in multiplications is attained by incorporating the scale factors of the final step of the algorithm into a quantization table.

The final scale factors of computation on each row cannot be incorporated into the quantization table because the scale factors are distinct for each coefficient. When elements are summed in the next phase, where the 1-D DCT is applied to each column, those scale factors cannot be factored out. It is important to note that if one optimizes the 2-D DCT calculation by incorporating necessary multiplications into the quantization matrix, the design no longer computes the DCT. It computes a version in which each coefficient needs to be scaled appropriately and is dependant on the presence of a quantization table. Thus, this 1-D DCT algorithm is optimized for use only within a compression core, such as JPEG, where quantization follows DCT computation. Since it is the intent of the project to have a stand alone DCT core, this optimization is not feasible.

It is worth noting that while the direct method of 2-D DCT calculation claims to reduce the number of multiplications in the row-column approach by a factor of two that is not always true. For example, when the Agui, Arai, and Nakajima 1-D DCT algorithm, optimized for use with a quantization table, is used in row-column decomposition, the direct method does not have as great a comparative savings. This is because the direct method must do some post processing after the 1-D DCT calculation stage so the constant scale factors cannot be incorporated into the quantization matrix. Thus with 13 multiplications per 1-D DCT computation (instead of 8 multiplications in the optimized version), 104 multiplications would result in the direct approach .Although the direct extension of the Arai, Agui, and Nakajima's 1-D DCT to two dimensions did not exactly halve the number of multiplications, it did reduce the number of multiplications by 40 with only a mere increase of 2 additions .The cost, however, is less regularity, which translates to greater complexity of control hardware.

The theoretical implementation of 1-D DCT algorithm is written in the form of simple transform given by

$$Y=AX, \text{ where X is 1-D array of data.}$$

Where it involves 64 multiplications and 56 additions to compute the entire the 1-D DCT and also it is very complex to route over FPGA, i.e. its butterfly architecture is very complex. The matrix A is given by

$$A=\begin{bmatrix} d & d & d & d & d & d & d & d \\ a & c & e & g & -g & -e & -c & -a \\ b & f & -f & -b & -b & -f & f & b \\ c & -g & -a & -e & e & a & g & -c \\ d & -d & -d & d & d & -d & -d & d \\ e & -a & g & c & -c & -g & a & -e \\ f & -b & b & -f & -f & b & -b & f \\ g & -e & c & -a & a & -c & e & -g \end{bmatrix}$$

3.1Chen's algorithm

The fast 1-D DCT algorithm that was selected for use in both the direct and row-column 2-D approaches was developed by Chen and Fralick .The 8-point, 1-D DCT, written in matrix factorization.

$$\begin{bmatrix} x_0 \\ x_2 \\ x_4 \\ x_6 \end{bmatrix} = \frac{1}{2}\begin{pmatrix} A & A & A & A \\ B & C & -C & -B \\ A & -A & -A & A \\ C & -B & B & -C \end{pmatrix}\begin{bmatrix} x_0+x_7 \\ x_1+x_6 \\ x_2+x_5 \\ x_3+x_4 \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ x_3 \\ x_5 \\ x_7 \end{bmatrix} = \frac{1}{2}\begin{pmatrix} D & E & F & G \\ E & -G & -D & -F \\ F & -D & G & E \\ G & -F & E & -D \end{pmatrix}\begin{bmatrix} x_0-x_7 \\ x_1-x_6 \\ x_2-x_5 \\ x_3-x_4 \end{bmatrix} \qquad (1)$$

Where A=cos (pi/4), B=cos (pi/8), C=sin (pi/8), D=cos (pi/16), E=cos (3*pi/16), F=sin (3*pi/16), G=sin (pi/16)

As mentioned earlier the 1-D Discrete Cosine transform can be expressed

as $$F(u)=\frac{1}{2}c(u)\sum_{y=0}^{7} f(x)\frac{\cos(2x+1)u\pi}{16} \qquad (2)$$

where c(u)= $\frac{1}{\sqrt{2}}$,u=0

1, otherwise.

In order to show how to use CORDIC algorithm for DCT, we give an example for F (0) and F(4).F(0) and F(4) can be expressed into (2) and (3),respectively.

$$F(0)=\{f(0)+f(7)+f(3)+f(4)\}\cos\left(\frac{\pi}{4}\right)+\{f(1)+f(6)+f(2)+f(5)\}\sin\left(\frac{\pi}{4}\right)$$
$$F(4)=\{f(0)+f(7)+f(3)+f(4)\}\cos\left(\frac{\pi}{4}\right)+\{f(1)+f(6)+f(2)+f(5)\}\sin\left(\frac{\pi}{4}\right) \qquad (3,4)$$

Where (2) and (3) are the rotation mode of CORDIC arithmetic. Therefore, in order to compute both F (0) and F (4), we need one CORDIC processor. Similarly, F (2) and F (6) can be obtained by using the rotation mode of CORDIC. For F (1) and F(7), F(5), and F(3), we need four CORDIC processors. Consequently, we can use six CORDIC processors for the 2D-DCT by applying the 1DDCT two times. In this paper, we modify the conventional DCT arithmetic flow so that the CORDIC (3) and CORDIC (6) processors have the same structure. Figure 1 shows the 8×1 DCT flow using the six –
CORDIC processors.

3.2 DCT using CORDIC architectures.

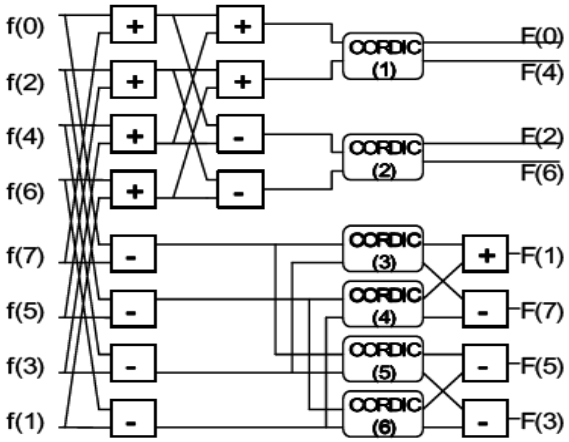


Fig3.1:1-D DCT architecture using CORDIC algorithm

The number of iterations can be reduced, since the coefficients for 8×1 DCT are fixed. Also, the compensation process for the final CORDIC calculation can be composed of adder and shifter without multiplier as expressed in (5).

$$X_{i+1} = X_i(1 + \gamma_i.F_i)$$
$$Y_{i+1} = Y_i(1 + \gamma_i.F_i) \qquad (5)$$

where γ_i has the value ± 1 and F_i is related to shift operation. Table 1 shows the detailed number of rotation for iterations and compensation in six CORDIC processors. In this project , the number of iterations is optimized within the precision requirements of IEEE Std. 1180-1990. The algorithm used by me in this project is given in the table 3.2..

processor	CORDIC(1)	CORDIC(2)	CORDIC(3)(6)	CORDIC(4)(5)
angle	$\frac{\pi}{4}$	$\frac{3\pi}{8}$	$\frac{7\pi}{16}$	$\frac{3\pi}{16}$
CORDIC iteration [σ,i]				
1	-1,0	1,0	1,0	1,1
2		1,2	1,1	1,3
3		1,3	1,3	1,10
4		1,6	1,10	1,14
5		1,7		
	Compensation iterations $[1+\gamma_i(i)F_i(i)]$			
1	$1-\frac{1}{4}$	$\frac{1}{2}+\frac{1}{8}+\frac{1}{64}$	$\frac{1}{2}+\frac{1}{8}$	$1-\frac{1}{8}$
2	$1-\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{8}+1$	$1+\frac{1}{64}$
3	$1+\frac{1}{256}$		$1+\frac{1}{4096}$	$1+\frac{1}{1024}$
4	$1+\frac{1}{512}$			$1+\frac{1}{4096}$
5	$1+\frac{1}{4096}$			

Table 3.2: Algorithm used for the computation of 1-D DCT.

Now we'll see the implementation of 2D DCT using the new CORDIC algorithm. Due to the inherent disadvantages of the conventional CORDIC algorithm like fixed number of iterations and the effect of the scaling factor, the conventional CORDIC algorithm lost its significance. But in this project results show that new CORDIC algorithm is not so efficient with respect to conventional CORDIC algorithm. The architecture for implementing 2D DCT using both CORDICs is same. Now we'll see the algorithm for the new Cordic algorithm is given in table 3.3

processor	CORDIC(1)	CORDIC(2)	CORDIC(3)(6)	CORDIC(4)(5)
angle	$\frac{\pi}{4}$	$\frac{3\pi}{8}$	$\frac{7\pi}{16}$	$\frac{3\pi}{16}$
CORDIC iteration[σ,i]				
1	[1,2]	[1,2]	[1,3]	[1,2]
2	[1,2]	[1,3]	[1,4]	[1,2]
3	[1,2]	[1,6]	[1,7]	[1,4]

4	[1,5]	[1,9]	[1,10]	[1,6]
5	[1,9]	[1,13]		[1,7]
6	[1,10]			[1,9]
7				[1,10]

Table 3.3: Algorithm of the new Cordic algorithm used for the Calculation of 1-D DCT.

The detailed description of both these algorithms is presented in chapter 2.

Chapter 4

CORDIC :AN ALGORITHM FOR VECTOR ROTATION

4.1Introduction to CORDIC algorithm

In the past decade, the unprecedented advances in VLSI technology have simulated great interests in developing special purpose, parallel processor arrays to facilitate real time digital signal processing. Parallel processor arrays such systolic arrays have been extensively studied. The basic arithmetic computation of these parallel VLSI arrays has often been implemented with a multiplication and accumulation unit(MAC) ,because these operations arise frequently in DSP applications. The reduction in hardware cost also motivated the development of more sophisticated DSP algorithms which require the evaluation of elementary functions such as trigonometric, exponential and logarithmic functions, which cannot be evaluated efficiently with MAC based arithmetic units. Consequently, when DSP algorithms incorporate these elementary functions, it is not unusual to observe significant performance degradation.

On The other hand, an alternative arithmetic computing algorithm known as CORDIC (Coordinate Rotation Digital Computer) has received renewed attention, as it offers a unified iterative formulation to efficiently evaluate each of these elementary functions. Specifically, all the evaluation tasks in CORDIC are formulated as a rotation of 2x1 vectors in various Coordinate systems. By varying a few simple parameters, the same CORDIC processor is capable of iteratively evaluating these elementary functions using the same hardware within the same amount of time. This regular unified formulation makes the CORDIC based architecture very appealing for implementation with pipelines VLSI array processors.

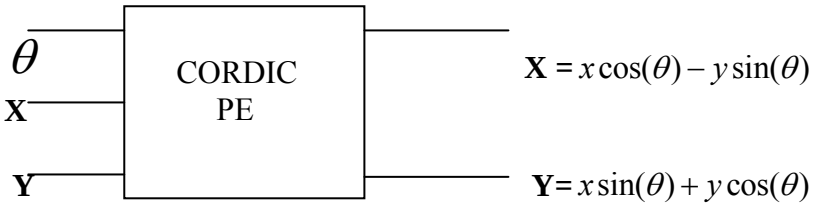
Typically CORDIC algorithm is used to implement linear transformations like DCT, DFT, Chirp-Z transform, DHT, digital filters like adaptive filters; Matrix based dsp algorithms like QR factorization, singular value decomposition.

The CORDIC is a class of hardware-efficient algorithms for the computation of trigonometric and other transcendental functions that use only shifts and adds to perform. The CORDIC set of algorithms for the computation of trigonometric functions was developed by Jack E. Volder in 1959 to help in building a real-time navigational system for the B-58 supersonic bomber. Later, J. Walther in 1971 extended the CORDIC scheme to other transcendental functions. The CORDIC method of functional computation is used by most

handheld calculators (such as the ones by Texas Instruments and Hewlett-Packard) to approximate the standard transcendental functions.

Depending on the configuration defined by the user, the resulting module implements pipelined parallel, word-serial, or bit-serial architecture in one of two major modes: rotation or vectoring. In rotation mode, the CORDIC rotates a vector by a specified angle. This mode is used to convert polar coordinates to Cartesian coordinates. For example consider the multiplication of two complex numbers x+jy and $(\cos(\theta) + j\sin(\theta))$ .The result u+jv, can be obtained by evaluating the final coordinate after rotating a 2x1 vector $[x \;\; y]^T$ through an angle θ and then scaled by a factor r.This is
accomplished in CORDIC via a three-phase procedure: angle conversion, Vector rotation and scaling.

The basic block diagram of CORDIC processing element is given below..



The basic conventional algorithm to rotate the coordinate axis is given below:

/* CORDIC angle Conversion */

Initialization $Z_0 = \theta$

For i=0 to b-1 Do

μ_i =sign(z_i) /* μ_i=1 *if* $Z_i > 0$

And

μ_i=-1 *if* $Z_i<0$ */

$$Z_{i+1} = Z_i - \mu_i \alpha_{m,i} ;$$

/* CORDIC Vector rotation */

Initialization $[X_0 \ Y_0]^T=[X \ Y]^T$

For i=0 to b-1 Do

$$x_{i+1} = x_i - m.\mu_i.y_i.\delta_{m,i}$$

$$y_{i+1} = y_i + \mu_i.x_i.\delta_{m,i}$$

/* Scaling Operation */

$$X_v = \frac{1}{k}X \quad , \quad Y_v = \frac{1}{k}Y$$

During the angle conversion phase, the angle θ is represented as the sum of a nonincreasing sequence of elementary rotation angles $\{\delta_{m,i} = 2^{-s_{m,i}}$ *defines a radix* 2 *number system.*, $0 \le i \le b-1\}$ where b is the number of rotations ,which in turn depends on the kind of precision we wanted, such that

$$\theta = \sum_{i=0}^{b-1} \mu_i \alpha_{m,i}$$

In the above algorithm, the set of parameters $\mu_i (= \pm 1)$ constitutes an implicit representation of θ, and b is the number of bits in the internal register. Variable $m \in \{1, 0, -1\}$ represents the rotation in three different systems: the circular, linear and hyperbolic respectively .In DCT, θ is known in advance and the operation is always in

Circular system only with m=1. The scaling factor $K = \prod_{i=0}^{b-1} \cos(\mu_i \tan^{-1} 2^{-i})$ will be a constant and independent of $|\mu_i|$=1.Hence K can be computed in advance and by calculating the limit comes to be around 0.60725.All the angles in angle conversion contains pre computation of arc tan. So this is implemented in the form of a look-up table in hardware. The trajectory of circular cordic rotation is given in figure below.

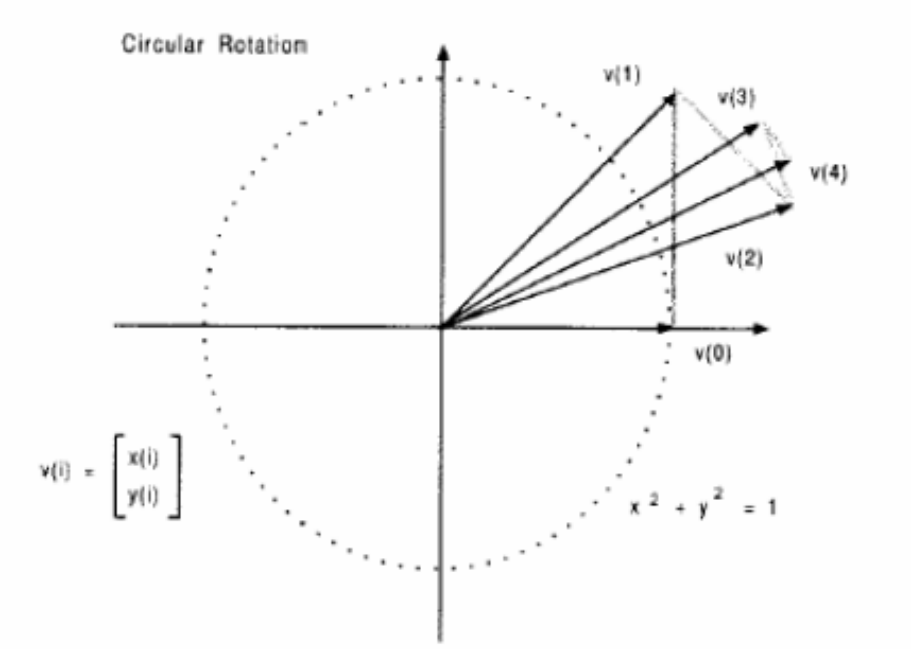


Fig4.1: Trajectory of circular Cordic rotation

The applications of CORDIC algorithm are given below.

4.2The Rotation Transform

All the trigonometric functions can be computed or derived from functions using vector rotations. The CORDIC algorithm provides an iterative method of performing vector rotations by arbitrary angles using only shift and add operations. The algorithm is derived using the general rotation transform:

$$\begin{aligned} X' &= X\cos(\phi) - Y\sin(\phi) \\ Y' &= Y\cos(\phi) + X\sin(\phi) \end{aligned} \qquad (1)$$

where (X', Y') are the coordinates of the resulting vector after rotating a vector with coordinates (X,Y) through an angle of ϕ in the Cartesian plane. These equations can be rearranged to give:

$$\begin{aligned} X' &= \cos(\phi).[X - Y\tan(\phi)] \\ Y' &= \cos(\phi).[Y + X\tan(\phi)] \end{aligned} \qquad (2)$$

Now, if the angles of rotation are restricted such that $\tan(\phi) = \pm 2^{-i}$ then the tangent multiplication term is reduced to a simple shift operation. Hence arbitrary angles of rotation can be obtained by performing a series of successively smaller elementary rotations. The

iterative equation for rotation can now be expressed as:

$$X_{i+1} = K_i(X_i - Y_i\sigma_i 2^{-i})$$
$$Y_{i+1} = K_i(Y_i + X_i\sigma_i 2^{-i}) \quad (3)$$

where $K_k = cos(tan^{-1}(2^{-k}))$ and $\partial_k = \pm 1$ depending upon the previous iteration. Removing the scale constant from the above equations yields a shift-add algorithm for vector rotation. The product K_k approaches the value of 0.6073. The CORDIC algorithm in its binary version can be expressed as a set of three equations as follows:

$$X_{k+1} = \left[X_k - mY_k.\partial_k.2^{-k}\right]$$
$$Y_{k+1} = \left[Y_k + X_k.\partial_k.2^{-k}\right] \quad (4)$$
$$Z_{k+1} = \left[Z_k - .\partial_k.\varepsilon_k\right]$$

Where $m = \pm 1$ and ε_k are prestored constants. The values of ε_k will become apparent from the following example for computing the sine and cosine functions.

4.3Computing Sine and Cosine Functions

To compute the *sin* θ and *cos* θ for $\theta \leq \pi/2$, we let m = 1, $\varepsilon_k = tan^{-1}(2^{-k})$ and define:

$$C = \prod_{k=0}^{n} \cos(\varepsilon_k) \quad (5)$$

Then the equations of the CORDIC algorithm for computing sine and cosine functions can be written down as:

$$X_{k+1} = \left[X_k - Y_k.\partial_k.2^{-k}\right]$$
$$Y_{k+1} = \left[Y_k + X_k.\partial_k.2^{-k}\right] \quad (6)$$
$$Z_{k+1} = \left[Z_k - .\partial_k.\varepsilon_k\right]$$

$\delta_k = \text{sgn}(Z_k)$, $X_0=C$, $Y_0=0$ and $Z_0 = \theta$ and *n* is the number of iterations performed. Then

$$X_{n+1} \approx \cos(\theta)$$
$$Y_{n+1} \approx \sin(\theta) \quad (7)$$

The well-known radix 2 system is considered here since it avoids the use of multiplications while implementing Equation (3). Hence a CORDIC iteration can be realized using shifters and adders/subtracters only. Figure 2 shows the structure of a processing element which implementing one CORDIC iteration. All internal variables are represented in a fixed number

of digits. The rotation angle θ is precalculated and can be stored in a register. The adders/subtracters are controlled by respectively $-m\mu_i, \mu_i \; and - \mu_i$ respectively.

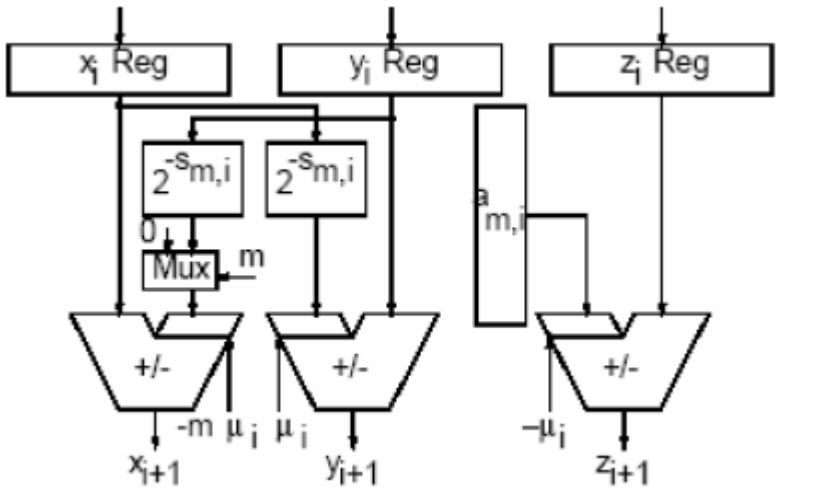


Fig 4.2:Basic structure of a processing element for one iteration

The rotation mode and vectoring mode are two control schemes for the CORDIC algorithm. In rotation mode, the objective is to rotate the given input vector (x , $y)^t$ with a desired/given angle θ. μ_i is equal to sign *(z,).* After n iterations, z_n is driven to zero and the total accumulated rotation angle is equal to θ. In vectoring mode, the desired rotation angle $\phi = \frac{1}{\sqrt{m}} \tan^{-1}(\sqrt{m}.\frac{y}{x})$ and magnitude $\sqrt{x^2 + m.y^2}$ are given for an input vector (x, y)^T. After n iterations, y_n is driven to zero, i.e. the objective rotates towards the x-axis. μ_i =-sign (x_i).sign (y_i). In this project, the CORDIC system is target to the rotation mode.

However, the CORDIC iteration is not a perfect rotation. Reference [7] points out that for a fixed-point implementation with data word length of W bits, no more than W CORDIC iterations need to be performed. The large number of iterations limits its speed performance seriously. Secondly, a scale factor operation is necessary after iteration in order to guarantee the final coordinate $[X_f, Y_f]^T$ the same m-norm as the initial coordinate [Xo ,Yo]^T has the same m-norm as the initial Co-ordinate $[X_o, Y_o]^T$. Besides, the algorithm computational accuracy needs to be taken into account as well. For example, the finite precision of the involved variables and the unavoidable rounding errors; a desired rotation angle θ can only be approximated by the n fixed rotation angles so that the accuracy achievable by the CORDIC rotation is determined by the last rotation angle $\alpha_{m,n-1}$.

The angles in design of DCT are known in advance. If conventional CORDIC is used, whatever is the angle, no of iterations are fixed. Instead of this, if we recode the angle in a proper way such that the sign sequence $\alpha_i = \sin^{-1} 2^{-i} \approx 2^{-i}$, instead of $\mu_i \in \{-1,1\}$, there is a possibility that several micro rotations will be skipped. This in turn reduces the number of iterations and speeds up the execution of CORDIC algorithm. In addition to this, every time to get the correct rotated coordinates, the pseudo result must be multiplied with a scaling factor K.If we eliminate this scaling block, the complexity of CORDIC algorithm will greatly reduced. And also, for a given rotation angle by modifying the micro rotation angles from $\alpha_i = \tan^{-1} 2^{-i}$ to $\alpha_i = \sin^{-1} 2^{-i} \approx 2^{-i}$, then we should not create a look up table and the burden of creating ROM will be greatly reduced. The large number of iterations limits its speed performance seriously and also consumes large power. Secondly, a scale factor operation is necessary in order to guarantee the final coordinate $[X_f, Y_f]^T$ has the same norm as the initial coordinate $[X_o, Y_o]^T$.

From the disadvantages of the Conventional CORDIC algorithm, the new CORDIC algorithm came out, where it overcomes all the disadvantages of CORDIC algorithm. This new CORDIC algorithm skips the operation of scaling factor correction as well as reduces the number of CORDIC iterations significantly. The algorithm is derived from the general rotation transform which shown in Equation (1). Since sine and cosine functions are included, they can be represented as following using *Taylor Series Expansion:*

$$\begin{aligned} \sin(\alpha) &= \alpha - (3!)^{-1}.\alpha^3 + (5!)^{-1}.\alpha^5 + \ldots \\ \cos(\alpha) &= 1 - (2!)^{-1}.\alpha^2 + (4!)^{-1}.\alpha^4 + \ldots \end{aligned} \tag{7}$$

The series up to third order is applied to Equation (1) with the correction of coefficient. The new CORDIC algorithm can be summarized below. Equation (8) describes a rotation together with a scaling of an intermediate plane vector $v_i = [x_i, y_i]^T$ *to* $v_{i+1} = [x_{i+1}, y_{i+1}]^T$.

For $\alpha_i = \sin^{-1} 2^{-i} \approx 2^{-i}$ with $i \in \{0, \ldots, n-1\}$, we have

$$
\begin{aligned}
x_{i+1} &= x_i.\cos(\alpha_i) - y_i.\sin(\alpha_i) \\
&= x_i.(1 - 2^{-1}.\alpha^2) - y_i.(\alpha - 2^{-4}.\alpha^3) \\
&= x_i.(1 - 2^{-2i-1}) - y_i.(2^{-i} - 2^{-3i-4}) \\
y_{i+1} &= y_i.\cos(\alpha_i) + x_i.\sin(\alpha_i) \\
&= y_i(1 - 2^{-1}.\alpha^2) + x_i.(\alpha - 2^{-4}\alpha^3) \\
&= y_i.(1 - 2^{-2i-1}) + x_i.(2^{-i} - 2^{-3i-4})
\end{aligned}
\tag{8}
$$

$$
\begin{aligned}
z_{i+1} &= z_i - \mu_i.\alpha_i \\
&= z_i - \mu_i.2^{-i} \qquad with\ \mu_i \begin{cases} 0\ z_i \le \alpha_i \\ 1\ z_i > \alpha_i \end{cases} and\ i \in \{0,....,n-1\}
\end{aligned}
$$

Since we quantize the micro-rotation angle $\alpha_i = \sin^{-1} 2^{-i}\ to\ 2^{-i}$, therefore, we should carefully choose the micro-rotation angles for CORDIC rotations. Table 2.1 shows the quantization error for the angles from $\alpha_0\ to\ \alpha_{10}$. The error is large at the beginning angles, e.g. $\alpha_0\ and\ \alpha_1$ (around 36% and 4.5% respectively), and decreases with the subsequent angles. Hence to achieve a high precision design, the CORDIC rotations should not start with large micro-rotation angles due to the large quantization error. Therefore the new CORDIC algorithm is modified to

$$
\begin{aligned}
&\alpha_i = 2^{-i} \\
&z_{i+1} = z_i - \mu_i.2^{-i}\ \ with\ i \in \{2,3,....,n-1\}
\end{aligned}
\tag{9}
$$

However, for large target rotation angle θ ,if α_i is set too small, the rotation precision will be increased as well as the number of CORDIC iterations required for this rotation angle. Hence the precision of the design needs to be traded of with the hardware limits and other performances as well.

i	Non-Quantized Micro-rotation Angle $\alpha_i = \sin^{-1} 2^{-i}$ (Radians)	Quantized Micro-rotation Angle $\alpha_i = 2^{-i}$ (Radians)	Quantization Error (%)
0	1.570796	1	**36.3380**
1	0.52360	0.5	**4.50726**
2	0.25268	0.25	1.06063
3	0.125328	0.125	0.26171
4	0.062541	0.0625	0.06556
5	0.031255	0.03125	0.01600
6	0.015626	0.015625	0.00640
7	0.0078126	0.0078125	0.00128
8	0.00390626	0.00390625	0.00026
9	0.00195313	0.00195313	0
10	0.00097656	0.00097656	0

Table 4.1: Shows the difference between the Conventional CORDIC and the new CORDIC algorithm.

The new CORDIC algorithm to the rotation angles θ from 0 to 45 degree(s). The value of the start point vector $[x,y]^T$ is $[1,0]^T$. Simulations matlab show that in maximum rotation number for each rotation angle is fixed to ten. Figure 2.4 shows the number of CORDIC iterations required to find value of the end point vector $[x,y]^T$ at each rotation angle. The micro-rotation start angle is set to a_2.

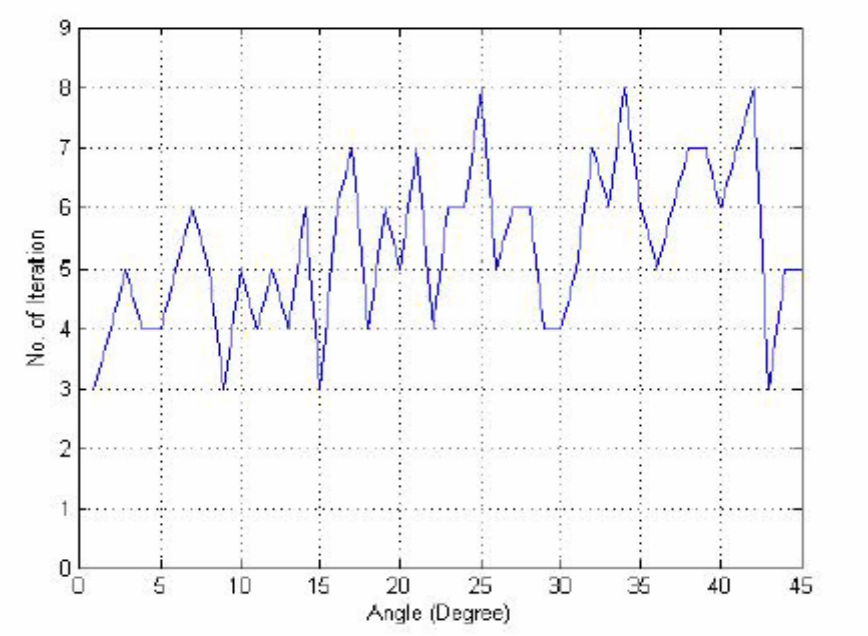


Fig 4.3: Number of Cordic iterations for input angle 0^o to 45^o

But in the case of the new CORDIC algorithm, the input angle is restricted to 0^o to 45^o.So if we want to have rotation for other angles less than 90^o, which we consider especially for 2D Discrete Cosine Transform, method of "domain folding" must be applied. For example if we want to rotate the angle less than 90^o and greater than 45^o, say θ, we have to perform $\phi = \frac{\pi}{2} - \theta$ .Then negative rotation of the θ must be performed so that

$$\begin{bmatrix} x'_+ \\ y'_+ \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

That means in the equations (8) we have to change '-'to '+' and vice versa. After computing the rotations, the final results are $x_{fd} = y'_+$, $y_{fd} = -x'_+$ .Eventhough we have approximated the angle as a mere right shifting, the error between the result computed by conventional CORDIC and the new CORDIC algorithm is very less which is shown in the error plot given below.

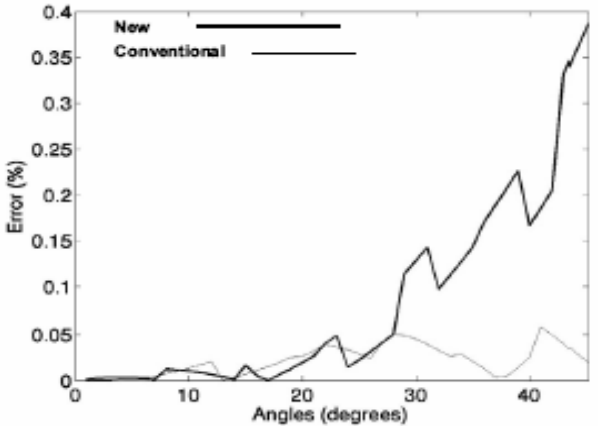


Fig 4.4: Error plot between New and Conventional CORDIC.

Even though the new algorithm is good in terms of lack of scaling factor and reducing the number of iterations, it has got some disadvantages like in the word length as the word length must be taken as 32 bits. It is the optimum word length. Now the architecture of each iteration of the new Cordic algorithm is given below.

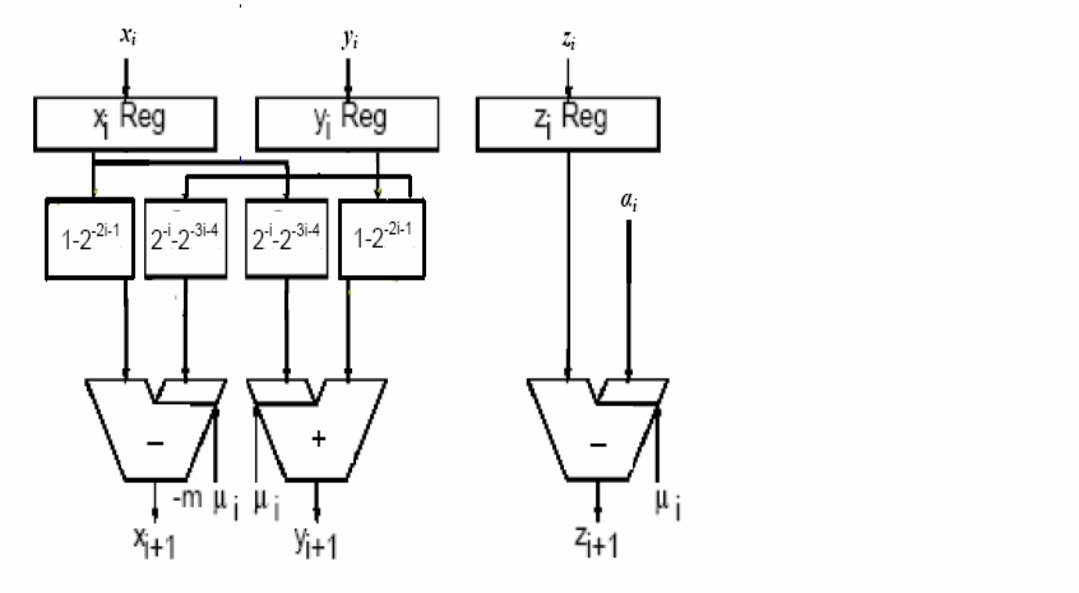


Fig 4.5: Architecture of a iteration in the new Cordic algorithm

The above blocks in between the adders and registers are shifters where the main shifting operation is 32 bit shift operation.

Chapter 5

Architectures of existing Cordic

There are three types of architectures with which the CORDIC ip core can be designed. They are 1) word-serial architecture

2) Parallel-Pipelined architecture.

3) Bit-Serial architecture.

Now the description of each and very architecture is described below..

5.1 Word-serial architecture:

Direct implementation of the CORDIC iterative equations (10) yields the block diagram shown in Fig 5.1. The vector coordinates to be converted, or initial values, are loaded via multiplexers into registers RegX, RegY, and RegA. RegA, along with an adjacent adder/subtractor, multiplexer, and a small arctan LUT, is often called an angle accumulator. Then on each of the following clock cycles, the registered values are passed through adders/subtractors and shifters. Every iteration takes one clock cycle, so that in *n* clock cycles, *n* iterations are performed and the converted coordinates are stored in the registers.

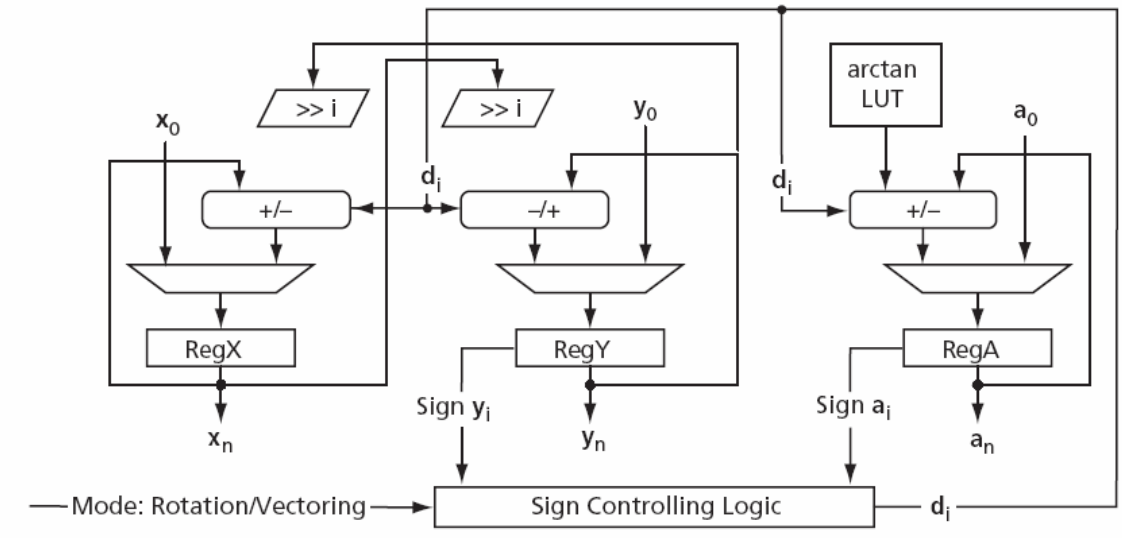


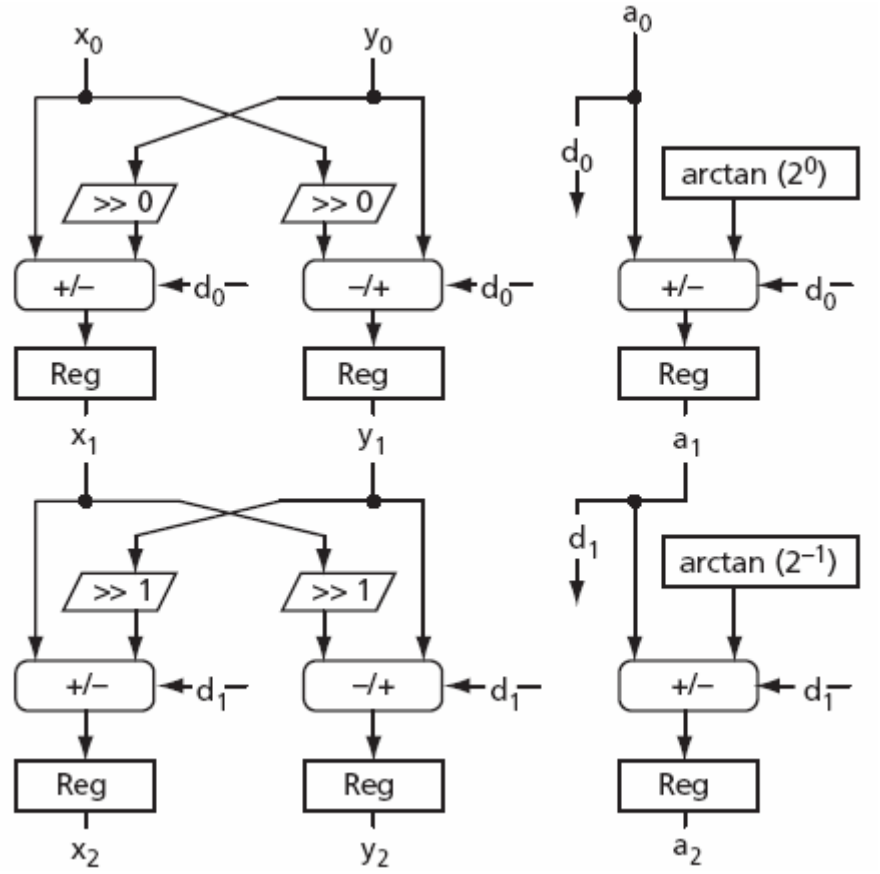
Fig 5.1: Word-serial CORDIC block diagram.

Depending on the CORDIC mode (rotation or vectoring), the sign-controlling logic watches either the RegY or the RegA sign bit. From, it decides what type of operation (addition or subtraction) needs to be performed at every iteration. The arctan LUT keeps a pre-computed table of the values. The number of entries in the arctan LUT equals the desirable number of iterations, *n.*

The word-serial CORDIC engine takes *n + 1* clock cycles to complete a single vector coordinate conversion.

5.2 Parallel-pipelined architecture:

This architecture presents an unrolled version of the sequential CORDIC algorithm above. Instead of reusing the same hardware for all iteration stages, the parallel architecture has a separate hardware processor for every CORDIC iteration. An example of the parallel CORDIC architecture configured for rotation mode is shown in fig 5.2



……..

……..

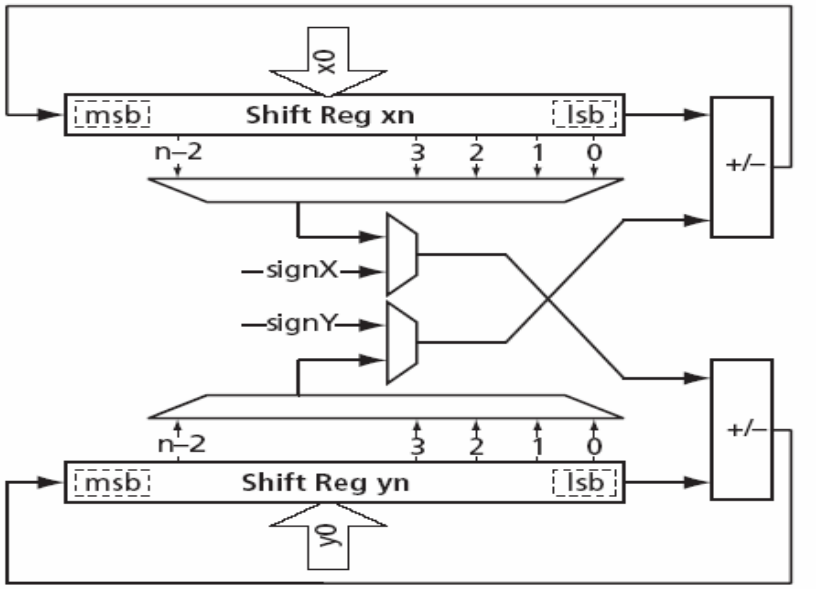
Fig 5.2: Parallel pipelined architecture

Each of the *n* processors performs a specific iteration, and a particular processor always performs the same iteration. This leads to a simplification of the hardware. All the shifters perform the fixed shift, which means these can be implemented in the FPGA wiring. Every processor utilizes a particular arctan value that can also be hardwired to the input of every angle accumulator. Yet another simplification is an absence of a state machine.

The parallel architecture is obviously faster than the sequential architecture described in the "Word-serial architecture "in fig 5.2. It accepts new input data and puts out the results at every clock cycle. The architecture introduces a latency of *n* clock cycles. The architecture which is used in the design of the 2D DCT is this parallel-pipelined architecture because this architecture not only provides high throughput but also results in low power consumption.

5.3Bit-serial architecture:

Whenever the CORDIC conversion speed is not an issue, this architecture provides the smallest FPGA implementation. For example, in order to initialize a Sine/Cosine LUT, the bit-serial CORDIC is the solution. Fig 3.3 depicts the simplified block diagram of the bit-serial architecture. The shift registers get loaded with initial data presented in bit-parallel form, i.e., all bits at once. The data then shifts to the right, before arriving the serial adders/subtractors. Every iteration takes *m* clock cycles, where *m* is the CORDIC bit resolution. Serial shifters are implemented by properly tapping the bits of the shift registers. The control circuitry (not shown in Fig 3.3) provides sign-padding of the shifted serial data to realize its correct sign extension. The results from the serial adders return back to the shift registers, so that in *m* clock cycles the results of iteration are stored in the shift registers.



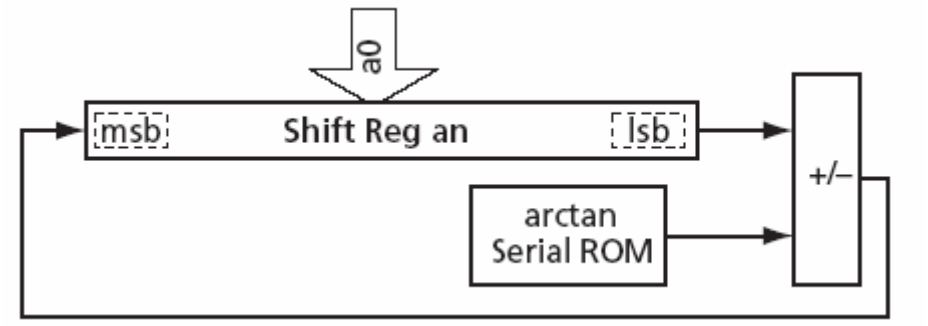


Fig 5.3: Bit-Serial CORDIC architecture.

5.4 Bit parallel iterative architecture:

The CORDIC structure as described in equations in chapter 4 ,is represented by the schematics when directly translated into hardware. Each branch consists of an adder-subtractor combination, a shift unit and a register for buffering the output. At the beginning of a calculation initial values are fed into the register by the multiplexer where the MSB of the stored value in the z-branch determines the operation mode for the adder-subtractor. Signals in the x and y branch pass the shift units and are then added to or subtracted from the unshifted signal in the opposite path.

The z branch arithmetically combines the registers values with the values taken from a lookup table (LUT) whose address is changed accordingly to the number of iteration. For n iterations the output is mapped back to the registers before initial values are fed in again and the final sine value can be accessed at the output. A simple finite-state machine is needed to control the multiplexers, the shift distance and the addressing of the constant values.

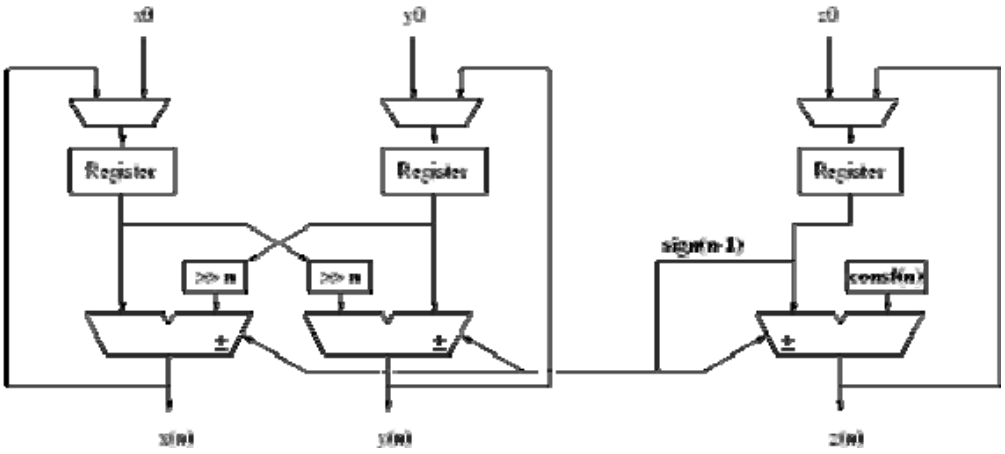


Fig 5.4:Bit parallel iterative architecture.

When implemented in an FPGA the initial values for the vector coordinates as well as the constant values in the LUT can be hardwired in a word wide manner. The adder and the subtractor component are carried out separately and a multiplexer controlled by the sign of the angle accumulator distinguishes between addition and subtraction by routing the signals as required. The shift operations as implemented change the shift distance with the number of iterations but those require a high fan in and reduce the maximum speed for the application In

addition the output rate is also limited by the fact that operations are performed iteratively and therefore the maximum output rate equals $\frac{1}{n}$ times the clock rate.

Thus the architectures of the CORDIC are mentioned and described. The architecture used in this project is the parallel pipelined architecture.

Chapter 6

FUNDAMENTALS OF LOW POWER DESIGN

Here we discuss ‘power consumption’ and methods for reducing it. Although they may not explicitly say so, most designers are actually concerned with reducing energy consumption. This is because batteries have a finite supply of energy (as opposed to power, although batteries put limits on peak power consumption as well). Energy is the time integral of power; if power consumption is a constant, energy consumption is simply power multiplied by the time during which it is consumed. Reducing power consumption only saves energy if the time required to accomplish the task does not increase too much. A processor that consumes more power than a competitor's may or may not consume more energy to run a certain program. For example, even if processor A's power consumption is twice that of processor B, A's energy consumption could actually be less if it can execute the same program more than twice as quickly as B.

Therefore, we introduce a metric: energy efficiency. We define the energy efficiency *e* as the energy dissipation that is essentially needed to perform a certain function, divided by the actually used total energy dissipation. The function to be performed can be very broad: it can be a limited function like a multiply-add operation, but it can also be the complete functionality of a network protocol. Note that the energy efficiency of a certain function is independent from the actual implementation and thus independent from the issue whether an implementation is low power.

It is possible to have two implementations of a certain function that are built with different building blocks, of which one has high energy efficiency, but dissipates more energy than the other implementation which has a lower energy efficiency, but is built with low-power components.

6.1 DESIGN FLOW

The design flow of a system constitutes various levels of abstraction. When a system is designed with an emphasis on power optimization as a performance goal, then the design must embody optimization at all levels of the design. In general there are three main levels on which energy reduction can be incorporated. The system level, the logic level, and the technological level. For example, at the system level power management can be used to turn off inactive modules to save power, and parallel hardware may be used to reduce global interconnect and allow a reduction in supply voltage without degrading system throughput. At the logic level asynchronous design techniques can be used. At the technological level several optimizations can be applied to chip layout, packaging and voltage reduction.

Low power design problems are broadly classified in to

1. Analysis
2. Optimization

Analysis: These problems are concerned about the accurate estimation of the power or energy dissipation at different phases of the design process. The purpose is to increase confidence of the design with the assurance that the power consumption specifications are not violated. Evidently, analysis techniques differ in their accuracy and efficiency. Accuracy depends on the availability of design information. In early design phases emphasis is to obtain power dissipation estimates rapidly with very little available information on the design. As the design proceeds to reveal lower-level details, a more accurate analysis can be performed. Analysis techniques also serve as the foundation for design optimization.

Optimization: Optimization is the process of generating the best design, given an optimization goal, without violating design specifications; an automatic design optimization algorithm requires a fast analysis engine to evaluate the merits of the design choices. A decision to apply a particular low power design technique often involves tradeoffs from different sources pulling in various directions. Major criteria to be considered are the impact on circuit delays, which directly translates to manufacturing costs. Other factors of chip design such as design cycle time, testability, quality, reliability, reusability; risk etc may all be affected by a particular design decision to achieve the low power requirement. The task of a design engineer is to carefully weigh each design choice with in specification constraints and select the best implementation.

Before we set to analyze or optimize the power dissipation of a VLSI chip, the basic understanding of the fundamental circuit theory of power dissipation is imminent. Further is the summary of the basic power dissipation modes of a digital chip.

6.2 CMOS COMPONENT MODEL

Most components are currently fabricated using CMOS technology. Main reasons for this bias is that CMOS technology is cost efficient and inherently lower power than other technologies.

The sources of energy consumption on a CMOS chip can be classified as

1. STATIC power dissipation, due to leakage current drawn continuously form the power supply and
2. DYNAMIC power dissipation, due to
   - Switching transient current,
   - Charging and discharging of load capacitances.

The main difference between them is that dynamic power is frequency dependent, while static is not. Bias (Pb) and leakage currents (Pl) cause static energy consumption. Short circuit currents (Psc) and dynamic energy consumption (Pd) is caused by the actual effort of the circuit to switch.

$$P = P_d + P_{sc} + P_b + P_l \quad \text{.....(2.1)}$$

The contributions of this static consumption are mostly determined at the circuit level. While statically-biased gates are usually found in a few specialized circuits such as PLAs, their use has been dramatically reduced in CMOS design. Leakage currents also dissipate static energy, but are also insignificant in most designs (less than 1%). In general we can say that careful design of gates generally makes their power dissipation typically a small fraction of the dynamic power dissipation, and hence will be omitted in further analysis.

6.2.1Dynamic power dissipation

Dynamic power can be partitioned into power consumed internally by the cell and power consumed due to driving the load. Cell power is the power used internally by a cell or module primitive, for example a NAND gate or flip-flop. Load power is used in charging the external loads driven by the cell, including both wiring and fan out capacitances. So the dynamic power for an entire chip is the sum of the power consumed by all the cells on the chip and the power consumed in driving all the load capacitances. During the transition on the input of a CMOS gate both *p* and *n* channel devices may conduct simultaneously, briefly establishing a hort from the supply voltage to ground. This effect causes a power dissipation of approx. 10 to 15%.

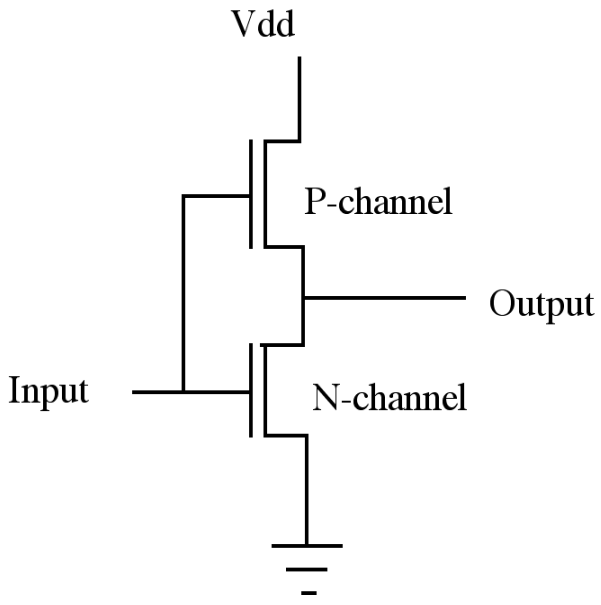


Figure 6.1: CMOS inverter.

The more dominant component of dynamic power is capacitive power. This component is the result of charging and discharging parasitic capacitances in the circuit. Every time a capacitive node switches from ground to Vdd an vice-versa energy is consumed. The dominant component of energy consumption (85 to 90%) in CMOS is therefore dynamic. A first order approximation of the dynamic energy consumption of CMOS circuitry is given by the formula:

$$P_d = C_{eff} V^2 f \quad \text{.....(2.2)}$$

where P_d is the power in Watts, C_{eff} is the effective switch capacitance in Farads, V is the supply voltage in Volts, and f is the frequency of operations in Hertz. The power dissipation arises from the charging and discharging of the circuit node capacitance found on the output of every logic gate. Every low-to-high logic transition in a digital circuit incurs a voltage change ΔV, drawing energy from the power supply. Ceff combines two factors C, the capacitance being charged/discharged, and the activity weighting α, which is the probability that a transition occurs.

$$Ceff = \alpha C.$$

6.2.2Short-Circuit Current In CMOS Circuit:

Another component of power dissipation also caused by signal switching called short-circuits power.

6.2.3Short-Circuit Current of an Inverter:

Figure shows a simple CMOS inverter operating at V_{dd} with the transistor threshold voltages of V_{tn} and V_{tp} as marked on the transfer curve. When the input signal level is above Vtn, the N-transistor is turned on; similarly, when the signal level is below V_{tp} the P-transistor is turned on. When the input signal V_i switches, there is a short duration in which the input level is Vtn and V_{tp} and both transistors are turned on. This causes a short circuit current from V_{dd} to ground and dissipates power. The electrical energy drawn from the source is dissipated as heat in the P and N transistors.

From the first order analysis of the CMOS transistors model, the time variation of the short- circuit current during signal transition is shown in the figure. The current is zero when

the inputs signal below V_{tn} or above V_{tp}. The current increase as Vi rises beyond Vtn and decreases as it approaches V_{tp}. Since the supply voltage is constant, the integration of the current over time multiplies by the supply voltage is the energy dissipated during the input transition period.

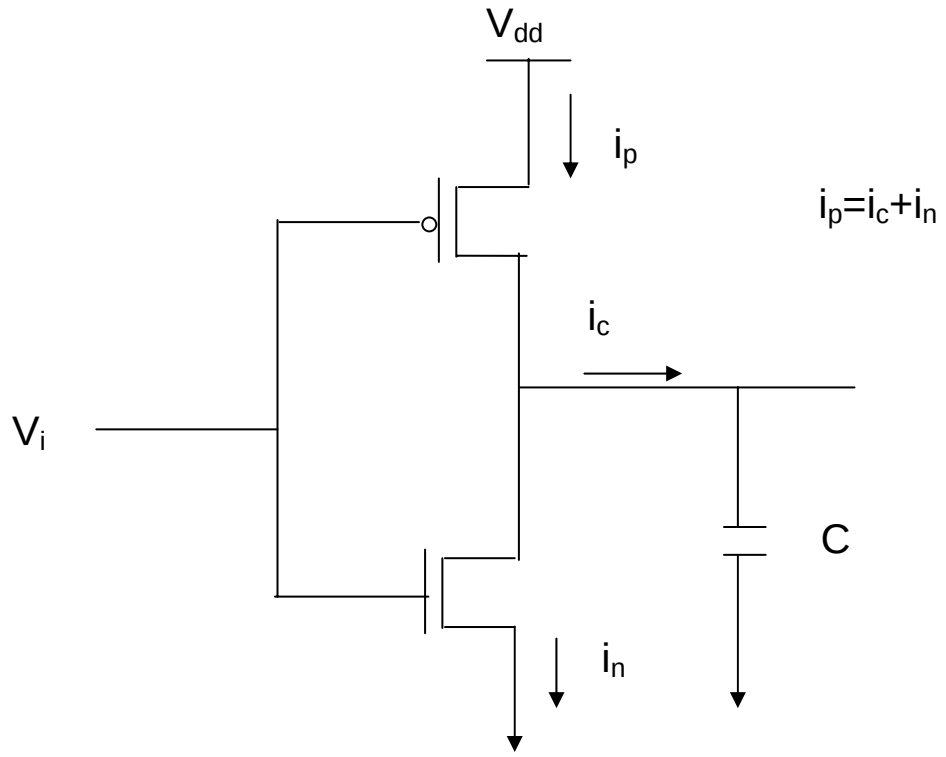


Fig 6.2: CMOS inverter and its transfer curve.

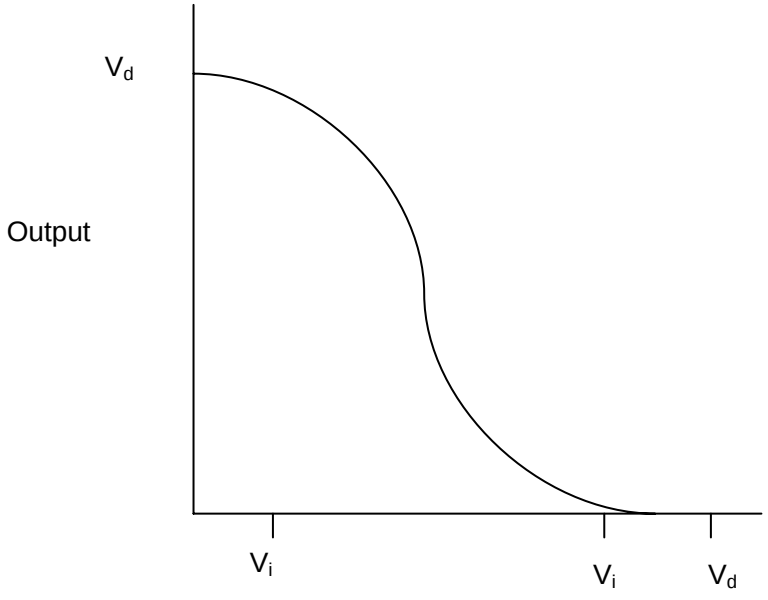


Fig 6.3: Transfer Characteristics of CMOS.

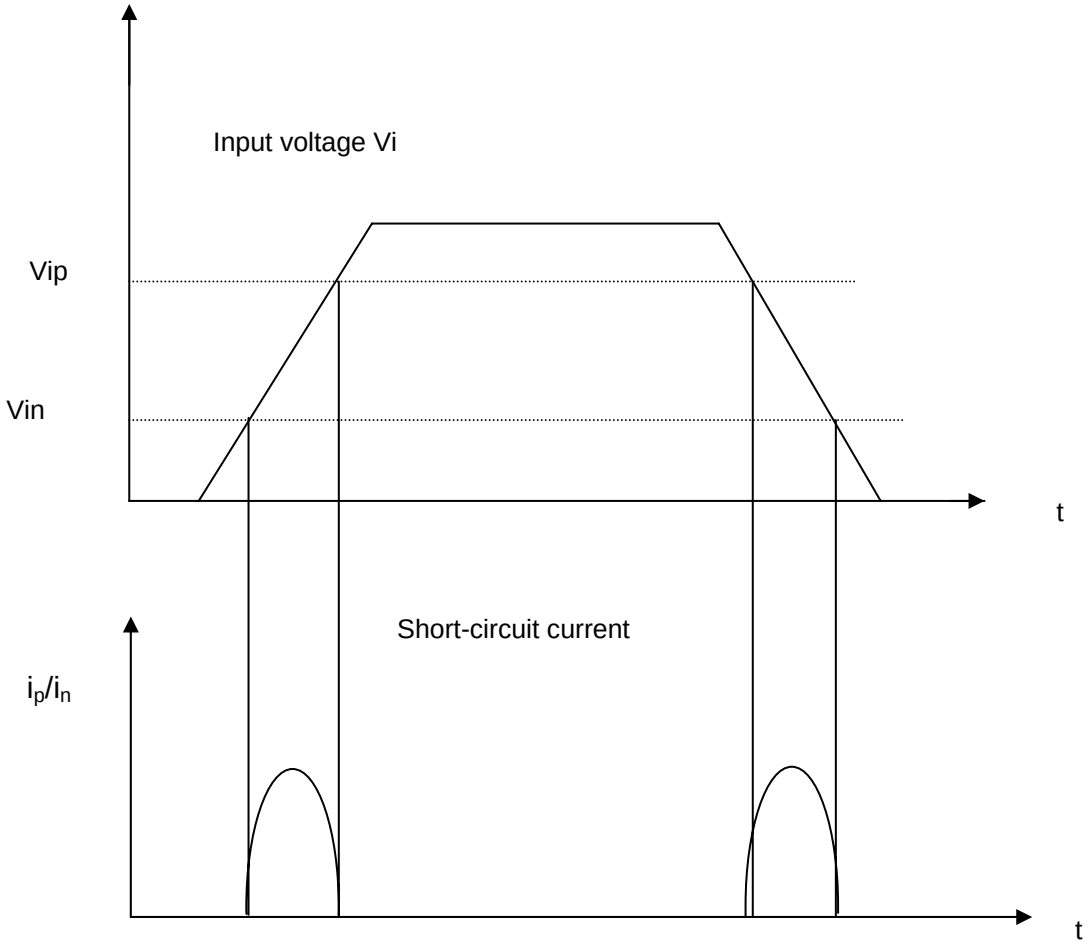


Fig 6.4 Short-circuit current of a CMOS inverter during input transition.

6.2.4 Static Power Dissipation

Strictly speaking, digital CMOS circuits are not supposed to consume static power from constant static current flow. All non-leakage current in CMOS circuits should only occur in transient when signals are switching. However, there are times when deviations from CMOS style circuit design are necessary.

An example is the pseudo NMOS logic. However, for special circuits such as PLAS or Register files, it may be useful due to its efficient area usage. In such circuits, there is tradeoff for power and area efficiency.

The pseudo NMOS circuit doesn't require a p-transistor network and saves half the transistors required for logic computation as compared to the CMOS logic. The circuit has a special property that the current only flows when the output is at logic 0.

When the output is at logic1, all the N-transistors are turned off and no static power is consumed,
Expect leakage current. This property may be exploited in a low power design. If a signal is known to have very high probability of logic1, say 0.99, it may make sense to implement the computation in pseudo NMOS logic. Conversely, if the single probability is very close to zero, we may eliminate the N- transistor network of a CMOS gate and replace it with a load transistor of N type.

An example where this future can be exploited is the system reset circuitry. The reset signal has extremely low activation probability (for example, during the power-on phase) which can benefit from such circuit technique. Other examples where single activation probabilities are extremely low are: test signals, error detection signals, interrupt signals and exception handling signals.

6.3 BASIC PRINCIPLES OF LOW POWER DESIGN

Conservation and trade-off are the philosophy behind most low power technique. The conservation school attempts to reduce power that is wasted with out a due course. The design skills required are in identifying, analyzing.

This often requires complex trade-offs decisions involving a designer's overall, intimate understanding of the design specifications, operating environment and intuition acquired from past design experience are keys to creative low power techniques.

It should be emphasized that no single low power technique is applicable to all situations. Design constraints should be viewed from all angles with in the bounds of the design specification. Low power considerations should be applied at all levels of design abstraction and design activities. Chip area and speed are the major trade-off considerations but a low power design decision also affects other aspects such as reliability, testability and design complexity. Early design decisions have higher impact to the final results and therefore, power analysis should be initiated early in the design cycle. Maintaining a global view of the power consumption is important so that a chosen technique does not impose restrictions on other parts of the system offset its benefits.

6.3.1 Reduce Voltage and Frequency

One of the most effective ways of energy reduction of a circuit at the technological level is to reduce the supply voltage, because the energy consumption drops quadratic ally with the supply voltage. For example, reducing a supply voltage from 5.0 to 3.3 Volts (a 44% reduction) reduces power consumption by about 56%. As a result, most processor vendors now have low voltage versions. The problem that then arises is that lower supply voltages will cause a reduction in performance. In some cases, low voltage versions are actually 5 Volt parts that happen to run at the lower voltage. In such cases the system clock must typically be reduced to ensure correct operation. Therefore, any such voltage reduction must be balanced against any performance drop. To compensate and maintain the same throughput, extra hardware can be added. This is successful up to the point where the extra control, clocking and routing circuitry adds too much overhead [58]. In other cases, vendors have introduced 'true' low voltage versions of their processors that run at the same speed as their 5 Volt counterparts. The majority of the techniques employing concurrency or redundancy incur an inherent penalty in area, as well as in capacitance and switching activity. If the voltage is allowed to vary, then it is typically worthwhile to sacrifice increased capacitance and switching activity for the quadratic power improvement offered by reduced voltage. The variables voltage and frequency have a trade-off in delay and energy consumption. Reducing clock frequency *f* alone does not reduce energy, since to do the same work the system must run longer. As the voltage is reduced, the delay increases. A common approach to power reduction is to first increase the performance of the module – for example by adding parallel hardware, and then reduce the voltage as much as possible so that the required performance is still reached. Therefore, major themes in many power optimization techniques are to optimize the speed and shorten the critical path, so that the voltage can be reduced. These techniques often translate in larger area requirements; hence there is a new trade-off between area and power.

6.3.2Reduce capacitance

Reducing parasitic capacitance in digital design has always been a good way to improve performance as well as power. However, a blind reduction of capacitance may not achieve the desired results in power dissipation the real goal is to reduce the product of capacitance and its witching frequency. Singles with high switching frequency should be routed with minimum parasitic capacitance to conserve power. Conversely, nodes with large parasitic capacitance should not be allowed to switch at high frequency. Capacitance reduction can be

achieved at most design abstraction levels: material, process technology, physical design (floor planning, placement and routing) circuit techniques, transistor sizing, logic restructuring, and architecture transformation and alternative computation algorithms.

6.3.3 Reduce Leakage and Static Currents

Leakage current, whether reverse biased junction or sub threshold current, is generally not very useful in digital design. However, designers often have very little control over the leakage current of the digital circuit. Fortunately, the leakage power dissipation of a CMOS digital circuit is several orders of magnitude smaller than the dynamic power. The leakage power problem mainly appears in very low frequency circuits or ones with “sleep modes” where dynamic activities are suppressed. Most leakage reduction techniques are applied at low level design abstraction such as process, device and circuit design. Memory chips that have very high device density are most susceptible to high leakage power.

Transistor sizing, layout techniques and careful circuit design can reduce static current. Circuit modules that consume static current should be turned off if not used. Sometimes, static current depends on the logic state of its output and we can consider reversing the signal polarity to minimize the probability of static current flow.

Chapter 7

DESIGN OF DCT CORE

The design flow of DCT core is shown particularly in the following flow-chart given in appendix A.The format of input word is different for different implementations where for example take the case of implementation of Chen's algorithm where the input word length is 8 bits. But in the case of implementation of CORDIC implementation, the input word is taken in the form 1QN format which if a fixed format. For that purpose all the data must be less than 1.So whatever is the data we have to make it less than 1 for Conventional CORDIC algorithm, where as for new CORDIC algorithm we have to make data in the form of 32 bits. CORDIC, as virtually any FPGA DSP core does, utilizes fixed-point arithmetic. In particular, the numbers the core operates with are presented as two's complement signed fractional numbers. To identify the position of a binary point separating the integer and fractional portions of the number, the Q format is commonly used. An mQn format number is an (n + 1)-bit signed two's complement fixed-point number: a sign bit followed by n significant bits with the binary point placed immediately to the right of the m most significant bits. The m MSBs represents the integer part, and (n-m) LSBs represent the fractional part of the number, called the mantissa. Table 4.1 depicts an example of a 1Qn format number.

Bit 2^n	Bit 2^{n-1}	Position of the Binary Point	Bits $[2^{n-2} : 2^0]$
Sign ↓	Integer bit		Mantissa

Table 7.1:1QN Format number.

Bit 2^n	Position of the Binary Point	Bits $[2^{n-1} : 2^0]$
Sign		Mantissa

Table 7.2: QN Format Number.

The following sections explain in detail the formats of the input and output signals. The linear and angular values are explained separately. The linear signals include Cartesian coordinates and a vector magnitude. These come to the CORDIC engine inputs x_o and y_o, or appear on its outputs x_n and y_n. Since the sine and cosine functions the CORDIC calculates are essentially the Cartesian coordinates of the vector, the angular signals include the vector phase that comes to the CORDIC engine input a_o, or appears on its output a_n. Both linear and angular signals utilize mQn formats and appropriate conversion rules from floating-point to the mQn formats.

7.1I/O linear Format

The CORDIC engine utilizes the 1Qn format shown in Table 3. Though the 1Qn format numbers are capable of expressing fixed-point numbers in the range from (-2^n) to ($2^n - 2^{m+n}$), the input linear data must be limited to fit the smaller range from (-2^{n-1}) to (2^{n-1}). In terms of floating-point numbers, the input must fit the range from -1.0 to1 .0. For example, the 1Q9 format input data range is limited by the following 10-bit numbers:

Max input negative number of -1.0:

1100000000 ⇔ 11.00000000

Max input positive number of +1.0:

0100000000 ⇔ 01.00000000

This precaution is taken to prevent the data overflow that otherwise could occur as a result of the CORDIC inherent processing gain. The output data obviously do not have to fit the limited range. To convert floating-point linear input data to the 1Qn format, follow the simple rule in EQ 10:

$$1Qn\ Fixed\text{-}Point\ Data = 2^{n-1} \times Floating\text{-}Point\ Data \quad (1)$$

Here it is assumed the floating-point data are presented in the range from -1.0 to 1.0. The product on the right-hand side of Eq (1) contains integer and fractional parts. The fractional part has to be truncated or rounded. shows a few examples of converting the floating-point numbers to the 1Q15 format.

To convert the 1Qn format back to the floating-point format, use EQ 11.

$$Floating\text{-}Point\ Data = 1Qn\ Fixed\text{-}Point\ Data/2^{n-1} \quad (2)$$

Suppose we are using the input in which one of the number is 34,then we have to convert to 1Q17 format where we have to divide 34 with 1000 and then we have to multiply with 2^16 and then round it…so that the end result is round(0.034*2^16) which results in 2228.Similarly the angle format for the CORDIC design is also the same, as mentioned above.

For the Design of 2D DCT using the New Cordic Algorithm, all the inputs must be 32 bits wide. All inputs are converted from decimal to fixed-point binary representation in Matlab. For this 32-bit design, the least 31 bits are used to represent the decimal fraction. The Most Significant Bit (MSB) is used as the sign bit. To check the output data of x' and y' at each rotation angle, we first convert angle θ from binary to decimal representation, and then

divided by 2^P, where P is the number of bits used to represent the decimal fraction. Then we can calculate the value of cos and sin θ for the estimation of output *x*' and *y*' respectively. If *x*' and *y*' are almost the same as the sine and
Cosine value that we calculate, then we can say the operation of the CORDIC ip is Correct. For example

$$(\theta)_2 = (00000010001110111110100011010100)_2 \ \textit{radians}$$
$$(\theta)_{10} = (37480660)_{10} / 2^p \ \textit{radians}$$
$$= (37480660)_{10} / 2^{31} = 0.0175 \ \textit{radians}$$

$$\cos(\theta) = \cos(0.0175) = 0.9998$$
$$(x')_2 = (01111111111110110011011100000100)_2$$
$$\therefore (x')_{10} = (2147200000)_{10} / 2^p = (2147200000)_{10} / 2^{31} = 0.9999 \approx \cos(\theta)$$

$$\sin(\theta) = \sin(0.0175) = 0.0175$$
$$(y')_2 = (00000010001011111111110001011100)_2$$
$$(y')_{10} = (36699228)_{10} / 2^p = (36699228)_{10} / 2^{31} = 0.0171 \approx \sin\theta$$

The above example verifies that output x' and y' are correct for its given rotation angle. Several outputs with different input rotation angles are chosen randomly to verify the correctness by following the steps illustrated in the example. Moreover, the simulation results are generated in waveforms so that we check not only the value of the outputs but also see if there is any timing matching problem within the overall design.

Now consider the design of the 2D DCT core with the Conventional Chen's algorithm. Every design in this project is based on row-decomposition algorithm and the architecture of every design is same which is mentioned below..

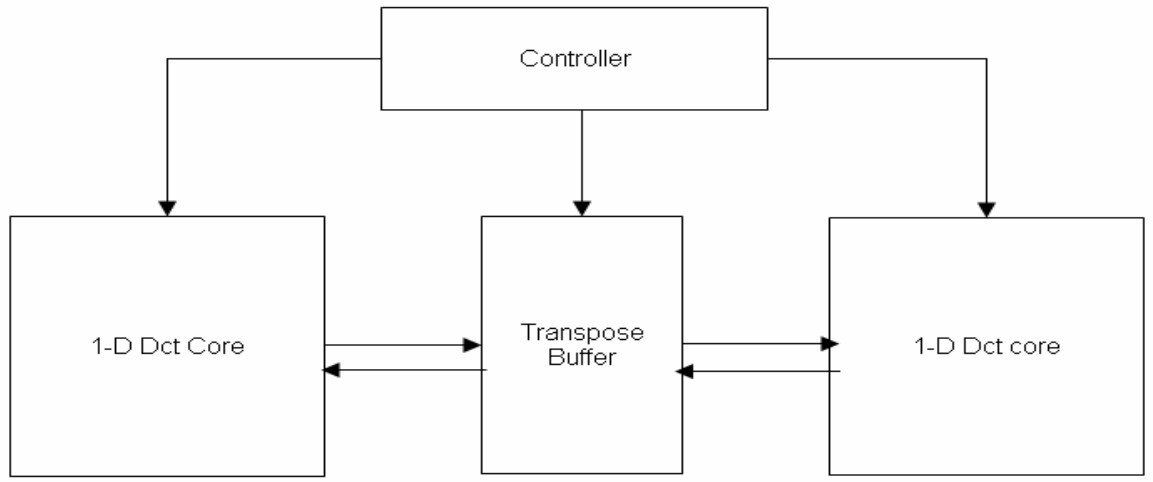


Fig-7.2: Architecture of 2D DCT used in this project

The basic architecture of each 2D DCT consists of the above blocks where the controller is the main part, which is a FSM ,especially a Mealy machine which gives status signals which are essentially signals which control the operation of 1-D DCT core and Transpose Buffer and output 1-D DCT core to work in synchronization. Transpose buffer is necessary block which performs the transpose of matrix which is an important block. The 1-D blocks in the block diagram which employs all the mentioned algorithms like Chen's algorithm, Cordic algorithms which is essentially an angle recoding (AR) algorithm. The specialty of the Angle recoded CORDIC algorithm where the angles in the computation of 2D DCT are predetermined and necessary iterations are calculated and the sign decision is also made. Due to this there is no need to incorporate the ROM in our design and numbers of iterations in our design are greatly reduced and thus results in reduced complexity.

But in the conventional DCT design using Chen's algorithm, the architecture is like a butterfly structure which is very difficult to map over FPGA, hence it takes a lot of area, and takes a lot of time for routing, i.e. complex routing. But even though we have used the Angle Recoded CORDIC algorithm, we have to do scaling which is again a unnecessary block if we used the new Cordic algorithm. So the new Cordic algorithm is used and the problem is solved, but the overhead lies in the input data width which should be 32 bits in minimum.

7.2 Design of controllers in DCT

The state diagram of the controller which is used in the design is given below…

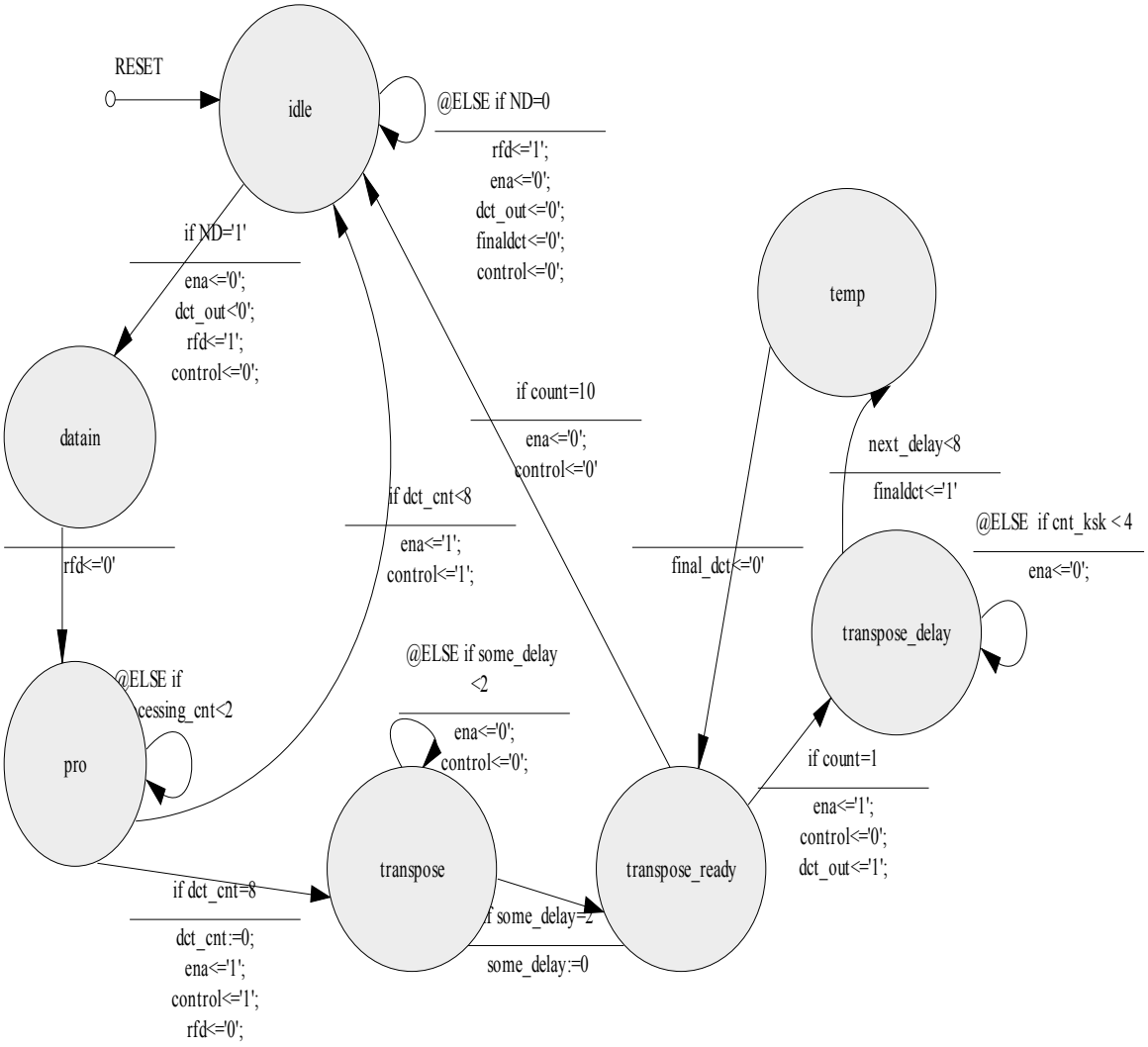


Fig 7.3: FSM for DCT design using Chen's algorithm

As mentioned in the above state diagram there are seven states namely idle,datain,processing,transpose,transpose_ready,transpose_delay,temp.The status signals are rfd,ena,dct_out,control,finaldct. Among the status signals the ena and control are the signals which control the transpose buffer. The main characteristic feature of transpose buffer is enabled, i.e it will start. The control signal controls the mode of transpose buffer. There are two modes in transpose buffer mode A and mode B. When the transpose buffer is in mode A, it will take all the date and stores in respective registers. When the same is in mode B, it will be transposing mode. The transpose buffer is absolutely pipelined.

But in the case of the Design of 2D DCT using AR Cordic and the new CORDIC algorithm is different and it is shown below.

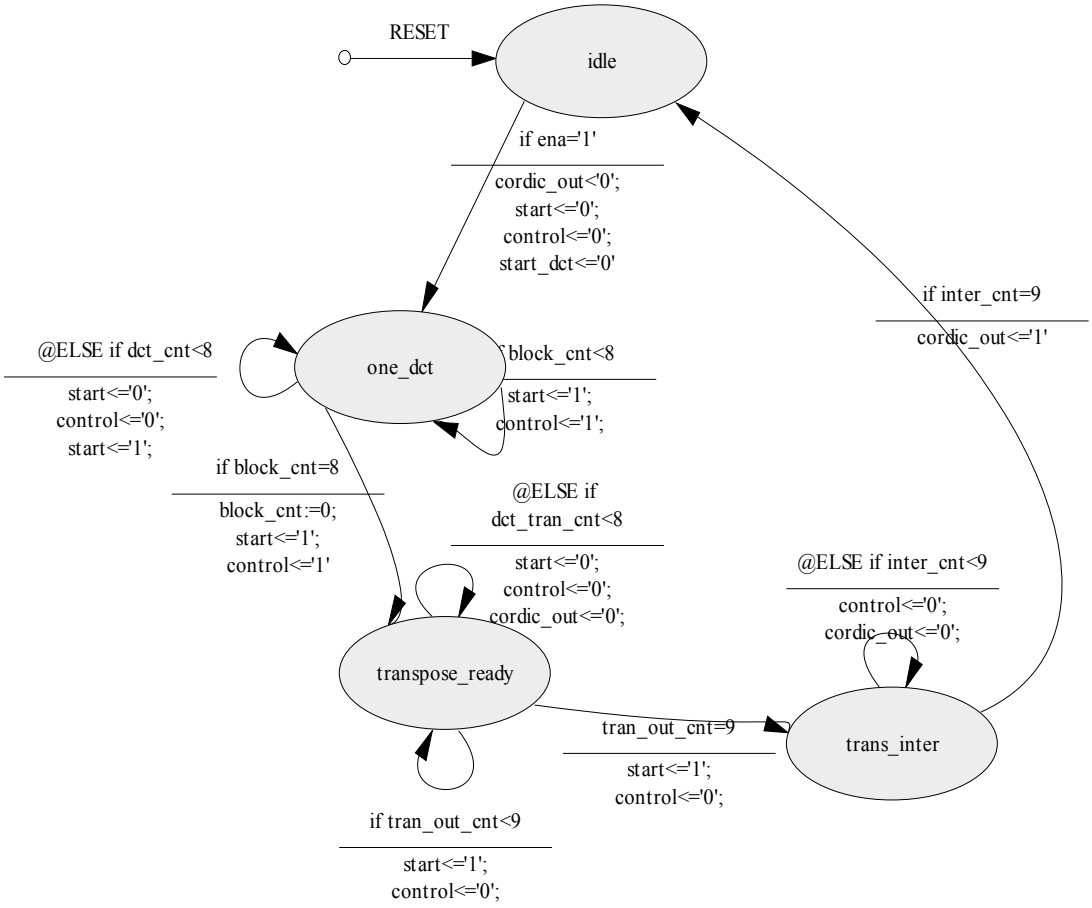


Fig 7.4 : FSM for DCT Design using CORDIC algorithm

In the design of DCT core using CORDIC algorithm, algorithms AR Cordic as well as the new CORDIC algorithm, the same state diagram is used. From the above state diagram it is implied the complexity of DCT architecture is less in the case of CORDIC architecture, where in the conventional one consists of multiplications, where in the normal CORDIC architecture consists of only rotations and shiftings. The sample code for the design of controller of 2D DCT using chen's algorithm and Cordic architecture. The VHDL codes of Controllers of DCT using chen's algorithm and controller of DCT using Cordic as well as the new CORDIC algorithm is given in appendix B&C.

7.3Design of transpose buffer

Now consider about the design of Matrix transposer cell which is necessary a some sort of transpose buffer. The need for real-time implementation of the transposition operation is felt particularly in image processing applications as they are dominated by matrix based techniques. For example, a wavelet operation on a two-dimensional array of data is executed as follows: First, the wavelet operation is executed on the rows (columns) of data followed by a transposition operation. This process is then repeated on the columns (rows) of

data. Note that in real-time processing environments; high data-processing rates are achieved using parallel and pipelined processors. Hence there is a great demand to speedup the execution of the transposition operation. A straightforward scheme is the memory addressing technique where the processed data are stored in rows (columns) and accessed in columns (rows) by altering the addressing sequence resulting the execution of the transposition operation. However, this technique limits the speed of processing because of

(i) The requirement for computation of the address sequence,

(ii) The need for storage space

(iii) The finite time for accessing the data.

Carlach *e/ a*] have presented a 8x8 DCT chip where a register based transposition stage is used. Panchanathan has proposed transposition architecture for real-time applications. Note that the last two architectures are primarily for serial transposition of data. The main drawback with all these implementations is the lack of modularity and cascadability which is required for VLSI implementation of large matrix transposition. The other drawback with these implementations is that most of these structures have complex communication, control and processor designs. Finally, some of the implementations are not suitable for real-time applications.

In this paper, we present a parallel and pipelined architecture for real-time MT This architecture is modular, cascadable and has a simple control design which makes possible FPGA implementation. The main advantages of the proposed architecture are as follows:

- It requires no address sequence computation or memory access time overheads.
- The execution time for transposition is very small and satisfies the real-time requirement for a variety of applications.
- The interconnections between the basic transposition cells are localized and hence the communication (U0 transfer) overhead is eliminated.
- There is a 100% utilization of the elements in the structure.

most importantly the structure is modular so that we can connect the required number of chips for any desired matrix size.

7.2.1Design of basic transpose cell*:*

The transposition of a matrix is a simple operation. Mathematically it can be expressed as :

$$X_{ij} = X_{ji} \quad For\ i,j = I,2,...,N$$

The above operation is illustrated for a 4 x 4 matrix (Figure 4.6)

$$\begin{bmatrix} X_{11} & X_{12} & X_{13} & X_{14} \\ X_{21} & X_{22} & X_{23} & X_{24} \\ X_{31} & X_{32} & X_{33} & X_{34} \\ X_{41} & X_{42} & X_{43} & X_{44} \end{bmatrix}$$

$$------> \begin{bmatrix} X_{11} & X_{21} & X_{31} & X_{41} \\ X_{12} & X_{22} & X_{32} & X_{42} \\ X_{13} & X_{23} & X_{33} & X_{43} \\ X_{14} & X_{24} & X_{34} & X_{44} \end{bmatrix}$$

Fig 7.5:Transposition of a Matrix.

The basic cell (transposer cell, shown in Figure 2) is designed to execute the transposition operation It has two modes of operation A and B which can be selected by a control signal C such that when

1. C=l A OUTPUT = A I"UT (A mode)

2 C=O B OUTPUT = B INPUT (B mode)

We note that K indicates the databus width The control signal is derived from the global clock signal Thus the communication is synchronous and the control is simple in structure.

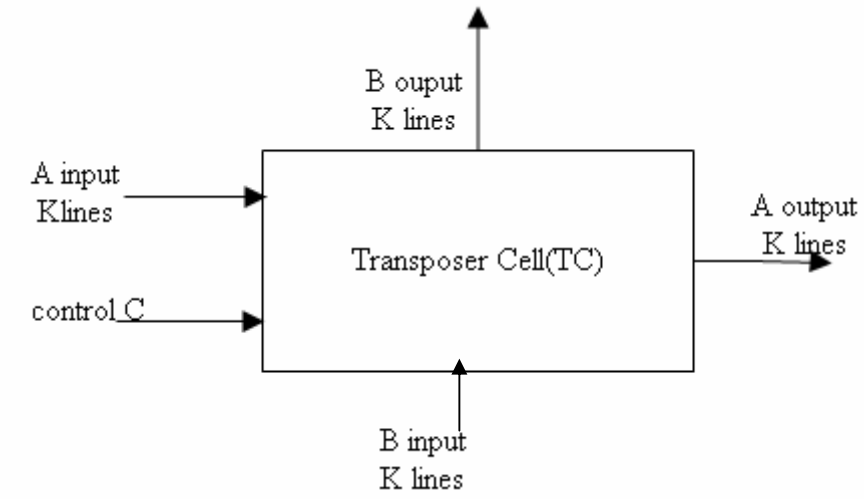


Fig 7.6: Transpose Cell

7.4Matrix transposition architecture

The Transposer Module (TM) essentially consists of N^2 basic cells interconnected as N rows of cells with each row consisting of N column of cells for the transposition operation on an N x N matrix.

For the sake of simplicity, the design of the architecture is illustrated for transposition of a 4x4 matrix. However, this concept is valid for any N x N matrix (Figure 2.7).An entire row (column) of data is loaded in and out of the module in each clock cycle. In the A-mode a row (column), initially the first row (column), of data of a matrix is loaded in

parallel into the cells TC1 1-TC41 at every clock cycle. Meanwhile, the second row (column) of data is prepared to be loaded into the TM. In the next clock cycle they are loaded into the cells TCII-TC41 while the first row (column) of data moves to the cells TC12-TC42 at the same time. The procedure is continued and at the end of four clock cycles all the rows (columns) of data are loaded into TM. As soon as the last row (column) of data is loaded into the cells TCl 1-TC41, the modules are switched to the B-mode of operation. In the next four clock cycles, the column (row) of data is drawn out of TM module through the outputs Al-A4 in the B-mode. Note that this output data is essentially the transposed version of the input matrix.

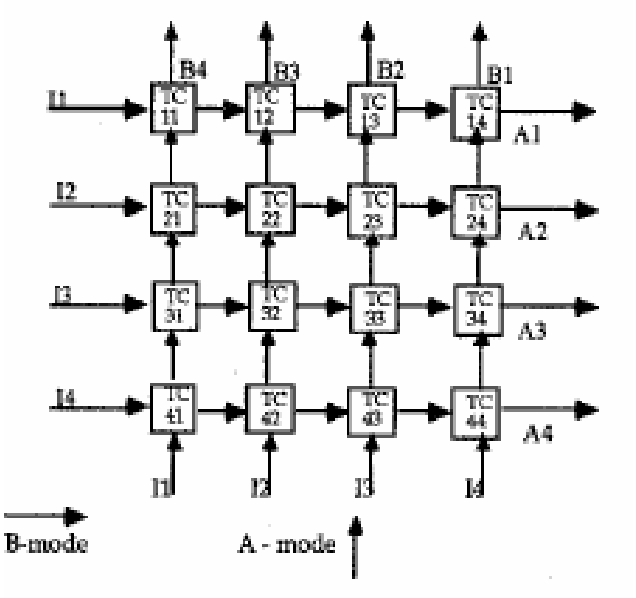


Fig 7.7 : Transpose Module

The external structure of the DCT core is given as follows:-

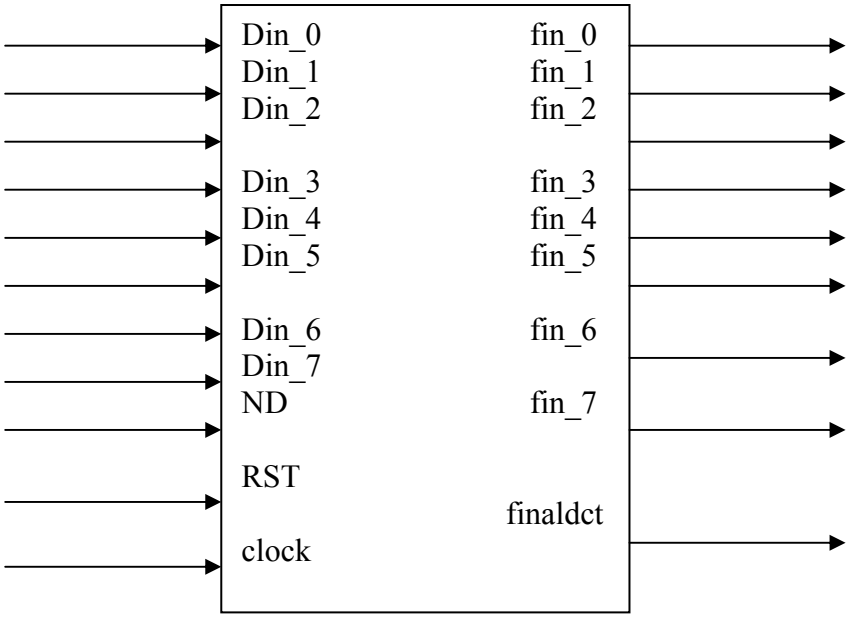


Fig 7.8 Architecture of DCT using Chen's algorithm

DIN_0…6(DATA INPUTS):

Din_0,Din_1,Din_2,Din_3,Din_4,Din_5,Din_6,Din_7 are inputs of 8 bits width .The Module will read the data when ND is high.

ND(NEW DATA):

When this input signal is high it indicates that valid data is available at the input DIN. If RFD is high then the module reads this data.

RST (Reset):

Reset allows user to restart the 2-D DCT process.

CLK (Clock):

This clock signal is used to synchronize the module and data input output operations

DOUT_0..6 (Data Output):

This output ports provides the results of 2-D DCT. When control signal finaldct is high,the DOUT_0,DOUT_1, DOUT_2, DOUT_3,DOUT_4,DOUT_5,DOUT_6,DOUT_7 is valid. The bit width of the outputs is 38.

FINAL DCT:

This signal indicates that whether the data at the output port is valid or not.

The controller is itself the main program which is the CORE DCT.The external structure of the DCT Core using CORDIC algorithm is given as follows:

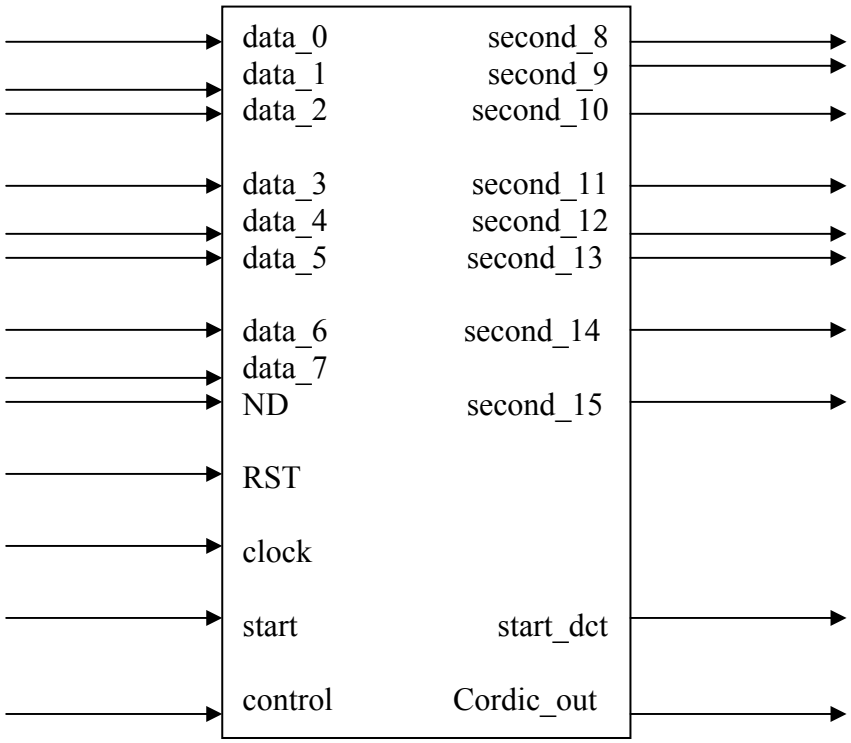


Fig 7.9 Architecture of DCT Core using CORDIC algorithm

DATA INPUTS:

Data_0,data_1,data_2,data_3,data_4,data_5,data_6,data_7,are 8 inputs which are of 17 bits in width .These will be fed to the CORE only when the ND is high.

ND:

When this input signal is high it indicates that valid data is available at the input DIN.

RST:

This is asynchronous reset where it resets the input without the intervention of clock.

START:

This is an inout signal where it denotes the occurrence of result form the 1-D DCT block. This in turn controls the transpose buffer.

CONTROL:

This is also an inout signal where it denotes the mode of transpose buffer. As previously notified that there are mode A and mode B. For first 8 rows of 8 elements each, control will be '0', after that control will be '1', where the transpose buffer transposes the matrix. This control is most important in the design.

STARTDCT:

This signal denotes the start of block for which the dct of that block is calculated.

CORDIC_OUT:

This signal denotes the presence of complete 2D DCT of the block which we presented as input.

DATA_OUTPUTS:

These outputs are also 17 bit wide which are valid only when Cordic_out is high, otherwise these can be ignored.

Chapter 8

SIMULATION RESULTS

The inputs to the DCT core using Chen's algorithm is given from a text file Named "srinew_1.txt" which contains the data as

34 34 34 33 34 29 35 33
34 34 34 33 34 29 35 33
34 34 34 33 34 29 35 33
34 34 34 33 34 29 35 33
34 34 34 33 34 29 35 33
36 36 30 27 33 31 31 32
32 32 35 30 32 34 31 28
31 31 27 29 30 31 28 29

Actually this data is the starting 8x8 matrix in a "lena512.bmp" file .Now we did the 2D DCT using matlab and got the result as

Y= [259.5000 4.7683 3.2404 -0.1992 0.2500 -0.5539 -4.5894 5.6385
7.9473 -0.7879 0.5547 -4.9323 1.9602 2.9784 -3.7971 3.3222
-5.0349 -0.2974 -1.5518 1.7250 -0.6765 -0.4525 1.8499 -2.2002
2.2619 1.1390 1.7039 0.9242 -0.7606 -1.3686 0.2143 1.1422
-1.0000 -1.1497 -0.3431 -1.3596 1.7500 1.1189 -1.4815 -0.6729
1.2107 0.4561 -1.8021 -0.1061 -2.0116 0.7097 1.6866 0.7552
-1.7028 0.2650 3.0999 1.6355 1.6332 -2.2546 -1.1982 -0.9011
1.3259 -0.4152 -2.3826 -1.5945 -0.8846 1.9693 0.5532 0.6540]

Now the same file is compiled, synthesized and simulated using Xilinx9.1ise and directly from the Xilinx itself we are saving the result in a file named "sri_res.txt".

Y_1= [259.4446 4.7681 3.2407 -0.1995 0.2499 -0.5531 -4.5885 5.6391
7.9477 -0.7878 0.5552 -4.9334 1.9602 2.9801 -3.7976 3.3228
-5.0341 -0.2975 -1.5525 1.7249 -0.6767 -0.4527 1.8498 -2.2003
2.2613 1.1392 1.7038 0.9248 -0.7611 -1.3692 0.2151 1.1424
-0.9998 -1.1495 -0.3428 -1.3595 1.7496 1.1189 -1.4814 -0.6731
1.2111 0.4561 -1.8030 -0.1065 -2.0117 0.7105 1.6866 0.7556
-1.7034 0.2647 3.0998 1.6358 1.6329 -2.2552 -1.1975 -0.9015
1.3269 -0.4151 -2.3829 -1.5951 -0.8847 1.9702 0.5528 0.6545]

Now the error between the original DCT calculated by Matlab and the one designed with the help of Chen's algorithm is given by:

Error= [0.0554 0.0002 -0.0003 0.0003 0.0001 -0.0008 -0.0009 -0.0006
-0.0004 -0.0001 -0.0005 0.0011 0.0000 -0.0017 0.0004 -0.0006
-0.0008 0.0001 0.0007 0.0000 0.0002 0.0002 0.0000 0.0001
0.0006 -0.0002 0.0001 -0.0006 0.0005 0.0005 -0.0008 -0.0002
-0.0002 -0.0002 -0.0002 -0.0001 0.0004 -0.0000 -0.0001 0.0002
-0.0004 -0.0000 0.0009 0.0005 0.0001 -0.0008 0.0001 -0.0004
0.0005 0.0003 0.0000 -0.0003 0.0003 0.0006 -0.0007 0.0003
-0.0010 -0.0002 0.0003 0.0006 0.0001 -0.0009 0.0004 -0.0005]

But for the CORDIC architecture which uses Cordic algorithm (Angle Recoded) which is 17 bits width and that too they are represented in 1Q16 format where the MSB is Non-decimal part and rest of the format which already mentioned in chapter -4.Now for that

purpose all the data given must be made to present in between -1.0 and 1.0.So each and every data must be multiplied with 2^16 and divided by 1000.We can also make a good approximation by multiplying with 2^6 if we make a 1000 as 1024 and made it 2^10.At the end also we can make the same approximation. This type of approximating enables us to read directly from without modifying the data. Otherwise we firstly modify the data using Matlab and save it in a file, later we can read from that text file with the help of VHDL test bench.

So doing the above modification to the above data, the text data is saved in "Sri_3.txt".The data which is present in that file is shown as:

2228 2228 2228 2163 2228 1901 2294 2163
2228 2228 2228 2163 2228 1901 2294 2163
2228 2228 2228 2163 2228 1901 2294 2163
2228 2228 2228 2163 2228 1901 2294 2163
2228 2228 2228 2163 2228 1901 2294 2163
2359 2359 1966 1769 2163 2032 2032 2097
2097 2097 2294 1966 2097 2228 2032 1835
2032 2032 1769 1901 1966 2032 1835 1901

Now the same file is read using dct core which uses CORDIC (Angle – recoding) algorithm, then we got the result as:

```
Y_2= [259.7198   4.6997   3.1128  -0.1831   0.2289  -0.6866  -4.6387   5.6000
       7.9193  -0.8545   0.5035  -5.0507   1.9379   2.9144  -3.8910   3.2806
      -5.0812  -0.3510  -1.6022   1.6785  -0.7324  -0.4730   1.8158  -2.2583
       2.3041   1.1902   1.7700   0.9766  -0.7324  -1.3275   0.2747   1.2207
      -1.0223  -1.1902  -0.3815  -1.3886   1.7700   1.1292  -1.4954  -0.7324
       1.1597   0.4883  -1.8005  -0.2136  -2.0599   0.7019   1.6174   0.7172
      -1.7090   0.2594   3.0823   1.6632   1.6479  -2.2583  -1.2054  -0.9155
       1.3275  -0.4120  -2.4261  -1.6479  -0.9003   1.9684   0.5798   0.6866].
```

The error between the original DCT calculated by matlab and the DCT CORDIC core is some what more than the Conventional DCT algorithm, but the improvement is there in terms of area and power consumption. The error is shown below..

```
Error= [  0.0198   0.0686   0.0276  -0.0161   0.0211   0.1328   0.0493   0.0385
          0.0280   0.0666   0.0512   0.1183   0.0224   0.0640   0.0938   0.0415
          0.0463   0.0536   0.0504   0.0465   0.0559   0.0205   0.0341   0.0581
         -0.0422  -0.0512  -0.0662  -0.0523  -0.0282  -0.0411-0.0604 -0.0785
          0.0223   0.0405   0.0384   0.0289  -0.0200  -0.0103   0.0139   0.0595
          0.0510  -0.0322  -0.0016   0.1076   0.0484   0.0078   0.0692   0.0380
          0.0061   0.0056   0.0176  -0.0277  -0.0147   0.0037   0.0072   0.0144
         -0.0016  -0.0033   0.0436   0.0534   0.0157   0.0010  -0.0266  -0.0327]
```

Now the same file is read with the DCT test bench, having a small modification.

To maintain the good precision, we have taken the input bit width as 45 .We can see the difference. So there is 1% error which can easily neglected. So with the help of the CORDIC algorithm even though there is an error, we can compensate this with low power and less area as well more compact design. This result is shown in the data results soon. Now considering the design with DCT using the New CORDIC algorithm. Since in the design in order to maintain a good precision, we have to take a more bit width. For example in the calculation for 3pi/8(chapter -1),the maximum value for i will be 13,means to have a proper precision, data atleast must be 50 bits wide, so automatically require more IOBs and in turn more area. Later we applied the DCT to the image and compared the PSNR of the three Lena images created by DCT using Chen's algorithm, DCT using "CORDIC algorithm" and later DCT using the "New Cordic Algorithm". Now consider the synthesis reports created by Xilinx 9.1ise.The family used for synthesizing is tabled below.

Device Family	Virtex2P
Device	XC2VP70
Package	FF1704
Speed	-6

Synthesis results show that the DCT using Chen's algorithm takes a lot of area, consume much power as there is a multiplier factor present in the algorithm, DCT using the new CORDIC algorithm is also inferior compared to the Angle Recoded Cordic algorithm in terms of area and power consumption. Also there must be trade off between required precision and the input bit width. For example to get the accurate precision upto 4 decimal digits after decimal point, input bit width is 43 bits in width, let it be 48 bits, which is a very large bit width considerable to Angle Recoded CORDIC Algorithm. Now Satisfying our need for good precision as well as less area, less power Angle Recoded CORDIC Algorithm is ow all the synthesis reports are given in tabular form.

Algorithm used	Area	Power Consumption(Xpower)
Conventional chen's algorithm	6.76% of total resources	261.43 mw is the peak power consumption
Angle recoded CORDIC	4.62% of total resources	210.20 mw of peak power consumption
The New CORDIC algorithm	5.67% of total resources	222.50 mw of peak power consumption

Table 8.1 : Simulation results

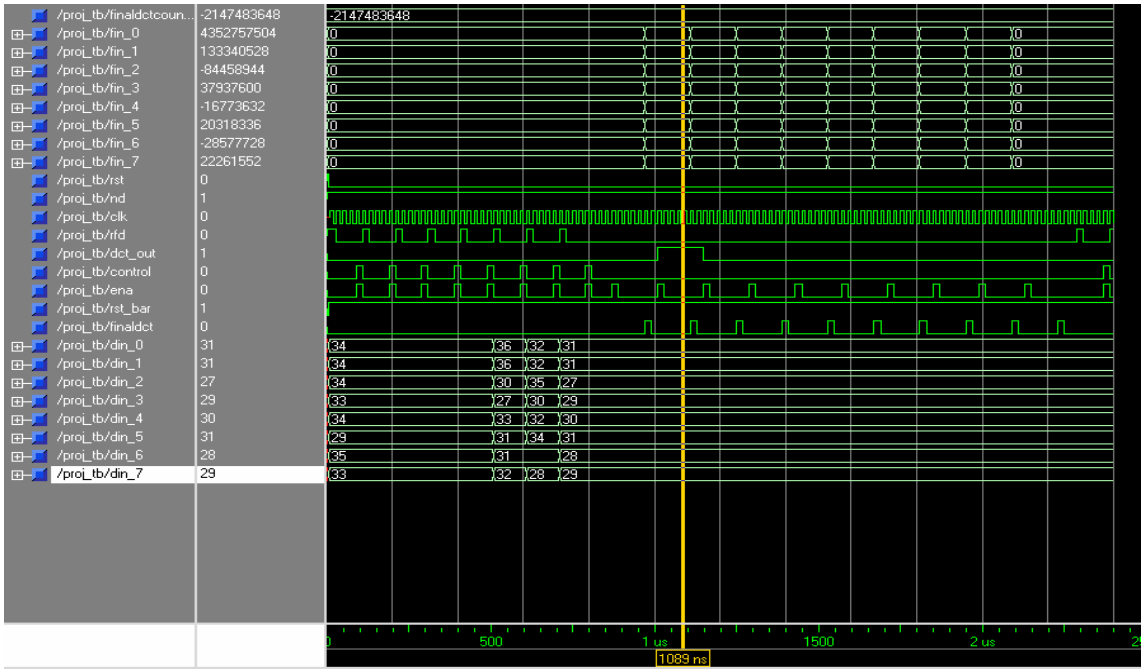


Fig 8.1:Timing Diagram showing the DCT results using the New CORDIC algorithm.

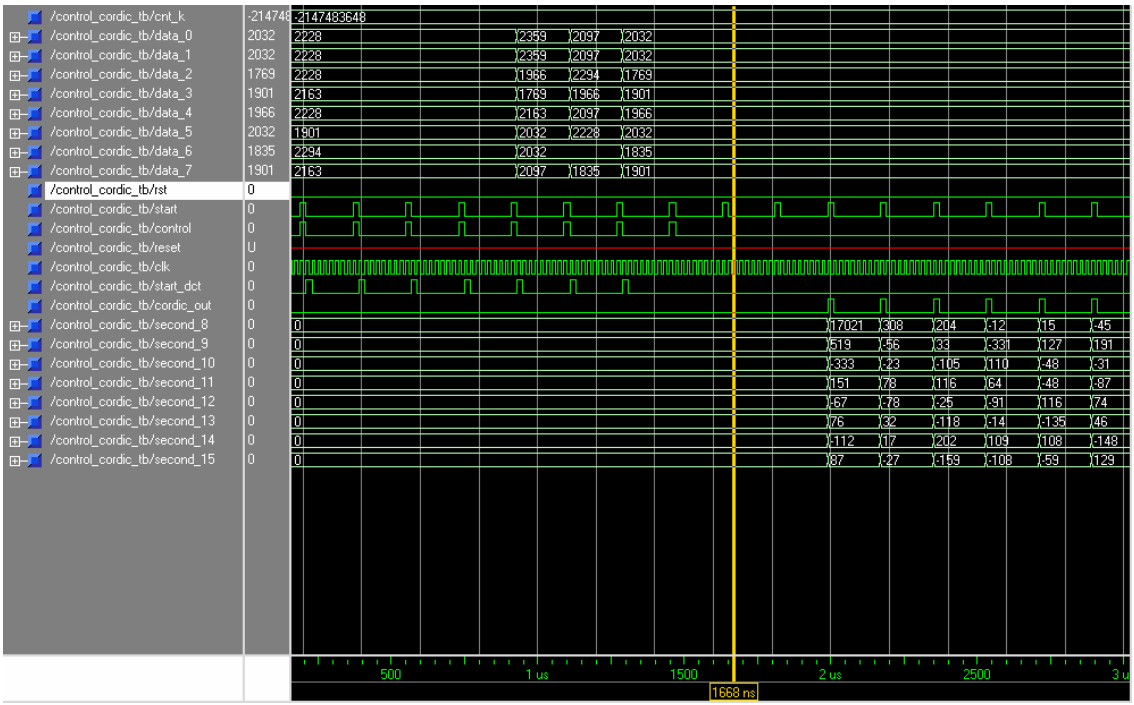


Fig 8.2:Timing diagram showing the DCT results using AR CORDIC algorithm

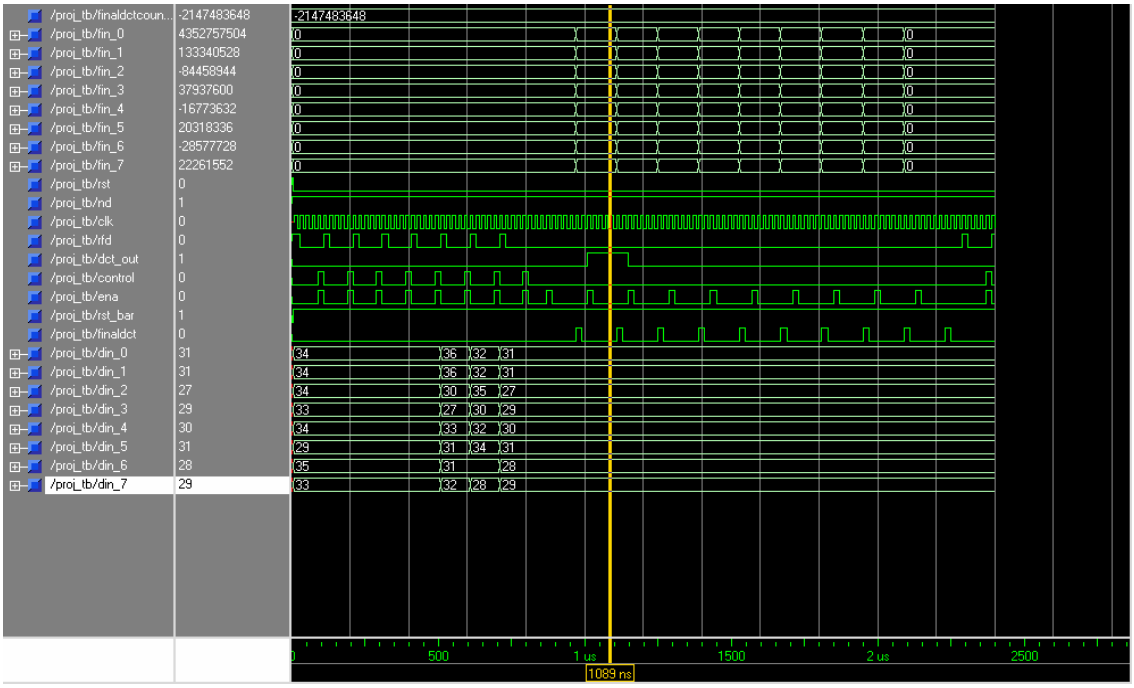


Fig 8.3:Timing diagram showing the DCT results using the New CORDIC algorithm.

From the above simulation results, we can say that DCT design using Angle Recoding algorithm is better compared with the Chen's algorithm and even the new CORDIC algorithm In terms of area and power consumption. But the design is not optimized in terms of time. So the of DCT using the Angle Recoded algorithm is better and it is very useful for the Modern DSP algorithms, since their implemenation always require less area and Low power.

Chapter 9

CONCLUSIONS

The Principal Contribution of this thesis is to modify the existing AR CORDIC algorithm so that the implementation of 2D Discrete Cosine Transform using AR CORDIC could be done with less complexity, consequently less area. So it is made as a hardware as well as performance efficient. Also a comparison with the new CORDIC algorithm is made. In concluding this thesis, the discussion of previous chapters is recapitulated in brief.

9.1Summary

The Discrete Cosine Transform is one of the most widely transform techniques in digital signal processing. In addition, this is also most computationally intensive transforms which require many multiplications and additions. Real time data processing necessitates the use of special purpose hardware which involves hardware efficiency as well as high throughput. Many DCT algorithms were proposed in order to achieve high speed DCT. Those architectures which involves multipliers ,for example Chen's algorithm has less regular architecture due to complex routing and requires large silicon area. On the other hand, the DCT architecture based on distributed arithmetic (DA) which is also a multiplier less architecture has the inherent disadvantage of less throughputs because of the ROM access time and the need of accumulator. Also this DA algorithm requires large silicon area if it requires large ROM size. Systolic array architecture for the real-time DCT computation may have the large number of gates and clock skew problem.

Digital signal processing (DSP) algorithms exhibit an increasing need for the efficient implementation of complex arithmetic operations. The computation of trigonometric functions, coordinate transformations or rotations of complex valued phasors is almost naturally involved with modern DSP algorithms. Popular application examples are algorithms used in digital communication technology and in adaptive signal processing. While in digital communications, the straightforward evaluation of the cited functions is important, numerous matrix based adaptive signal processing algorithms require the solution of systems of linear equations, QR factorization or the computation of Eigen values, eigenvectors or singular values. All these tasks can be efficiently implemented using processing elements performing vector rotations. The Coordinate Rotation Digital Computer algorithm (CORDIC) offers the opportunity to calculate all the desired functions in a rather simple and elegant way.

The DCT based on CORDIC algorithm doesn't need multipliers .Morever; it has regularity and simple hardware architecture, which makes it easy to be implemented in VLSI. Also, the CORDIC based DCT algorithm can support the high performance applications such as HDTV due to high throughput. In this thesis, DCT using normal Chen's algorithm was

implemented with word serial architecture, using AR Cordic algorithm and the new CORDIC algorithm was implemented. The architecture with which the DCT Core has been implemented was also described. The state machine for the controller part is also mentioned ,correctly drawn and the code was also provided. The AR CORDIC algorithm with which the DCT is implemented was also described. The Architectures for the AR Cordic algorithm as well as the new CORDIC algorithm are the same.

In the new AR CORDIC algorithm, as per the rule the CORDIC algorithm involves a lot of iterations N, as usually more number of iterations means more precision and accurate results. The Conventional CORDIC algorithm states that, after shifting and rotation process is done, a scaling process is also there which involves the multiplication of the result with 0.60725.So if we want to reduce the complexity of CORDIC algorithm; we have to concentrate on reducing the number of iterations and also in removing the scaling factor. So these factors are utilized by AR CORDIC algorithm and The New Cordic Algorithm respectively. In this ref [4], states an algorithm which utilizes an effective algorithm, but in this thesis work, that same algorithm is modified and presented which is working perfectly according to the simulation results. The modification lies in the Angle recoding and also in the changing of the scaling factor which also involves the same shifting and adding without much error. In DCT we already pre-determined the angles with which we have to rotate, so the angle recoding is easily possible. Also the scaling is different for different angles. The algorithm is briefly explained in chapter [2].The resulting architecture is simulated in Modelsim 5.7f, by taking an 8X8 matrix in the form of a text file. The detailed simulation are given in chapter [8].In the new Cordic algorithm, to put a sufficient precision of 0.0001 around we have to take the input bit width of around 50 bits which is quite a large value. So from the simulations, the new algorithm is not good in terms of area and power consumption. But there is a tradeoff in terms of accuracy and in put bit width. If we sacrifice this we can get good results.By seeing the synthesis results which are given in chapter [8], the AR CORDIC algorithm is better in terms of area and power consumption.

9.2Future Work

There are several issues regarding this CORDIC implementation of DCT are not issued in this thesis. The main issue of its kind is timing issues. The area and issues are not optimized for any of the three designs. This is due to lack of knowledge on Synopsys design compiler. The hardware implementation of this DCT Core isn't made. If it was done then we would have estimated the correct use of then DCT and verify its use in some real time implementations.

References

[1]Volnei A.Pedroni.CRICUIT DESIGN WITH VHDL. New Delhi: Prentice-Hall of India, 2004

[2]Stefan Sjoholm and Lennart Lindh. VHDL FOR DESIGNERS. Singapore: Prentice-Hall, 1995.

[3]Keshab K.Parhi .VLSI Digital signal processing systems.Canada: Wiley, 1999.

[4]Hyeonuk Jeong, Jinsang Kim and Won-Kyung Cho, "Low –Power Multiplier less DCT Architecture Using Image Data Correlation". IEEE Transactions on Consumer Electronics. Volume 50, No.1,(February 2004) P.262-266.

[5]J.E Volder, "The CORDIC trigonometric computing technique", IRE Transactions on Electronic computers. Volume EC-8, No 3, (September 1959) P.330-334.

[6]J.S Walther, "A Unified algorithm for elementary functions" AFIPS spring joint computer conference. Volume 38 (May 1971) P.379-385.

[7]Ray Andraka: A Survey of CORDIC Algorithms for FPGA based Computers. "www.fpga-guru.com/files/crdcsrvy.pdf" (online).info@andraka.com.

[8]Helmut Knaust: How do Calculators Calculate?. "www..math.utep.edu/Faculty/Helmut/wcordic.html " (online).HKnaust@texas.edu.

[9] Yu Hen Hu, "CORDIC Based VLSI-Architectures for Digital Signal Processing"; IEEE Signal Processing Magazine, (July 1992) P.16–35.

[10] Ruiqi Zhang, Jong Hun Han, A.T.Erdogan, T.Arslan. "Low Power CORDIC IP core implementation", ICASSP 2006 proceedings. Volume 3(July 2006) P.956-959.

[11]George S. Taylor and Gerard M. Blair, "Design of Discrete Cosine Transform in VLSI" , (September 1997) P.1-14.

[12]O.Fatemi and S.Panchanathan. "VLSI architecture of a scalable Matrix Transposer." Innovative systems in silicon conference(June 1996),P382-390.

[13]Yu Hen Hu and Zhenyang Wu. "An Efficient CORDIC Array Structure for the implementation of Discrete Cosine Transform", IEEE Transactions on signal processing , Volume 43(January 1995),P331-336.

[14]Ling Wang and Liro Hartimo. "Systolic Architectures for computing 2-D DCT Based on CORDIC Techniques", IEEE Transactions on signal processing, Volume 6(1994) ,P73-75.

[15]ACTEL (March,2006)."www.actel.com/coreCORDIC CORDICRTLGenerator".(Online)

[16]Ken Cabeen and Peter Gent. "Image Compression and Discrete Cosine transform", Tutorial presented at College of Redwoods, P1-11.

[17]H.Dawid, H.Meyr, “Chapter 24 CORDIC Algorithms and Architectures,” available from: http://www.google.com/url?sa=U&start=1&q=http://www.cad.eecs.berkeley.edu/~newton/Classes/EE290sp99/lectures/ee290aSp996_1/cordic_chap24.ps&e=9777.

[18]Vincent Cosidine.“CORDIC Trigonometric Function Generator for DSP”, IEEE Trans-Ctions on signal processing,1989,P2381-2384.

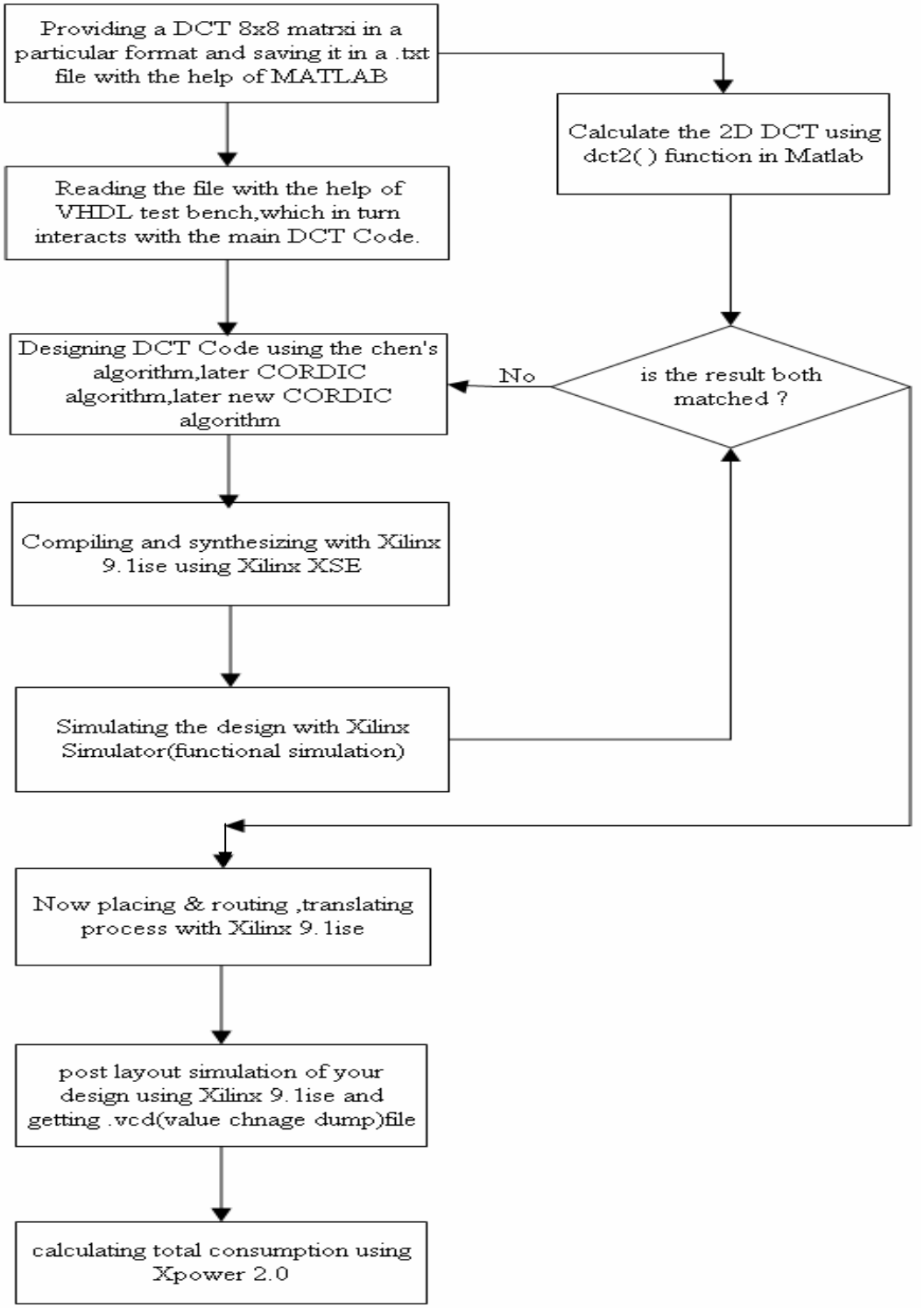
[19]Hassan M.Ahmed and Kin-Ho Fu. “A VLSI Array CORDIC Architecture”, IEEE Transactions,1989 P2385-2388.

[20]Y.H.Hu and S.Naganathan , “An angle Recoding algorithm for CORDIC algorithm implementation,” IEEE Transactions on computers, Volume 42(January 1993) P99-102.

[21]Syed Ali Khayyam, “The Discrete Cosine Transform (DCT): Theory and Applications”, Tutorial, Dept. of Electrical and Computer Engineering, Michigan State University (March 2003)

Appendix A

Flow chart for Design flow for DCT design



Appendix B

VHDL program for Controller in the DCT using Chen's algorithm

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;


entity control_dct is
port ( DIN_0: in std_logic_vector(7 downto 0);
       DIN_1: in std_logic_vector(7 downto 0);
       DIN_2: in std_logic_vector(7 downto 0);
       DIN_3: in std_logic_vector(7 downto 0);
       DIN_4: in std_logic_vector(7 downto 0);
       DIN_5: in std_logic_vector(7 downto 0);
       DIN_6: in std_logic_vector(7 downto 0);
       DIN_7: in std_logic_vector(7 downto 0);
       ND,RST,CLK: in std_logic;
       RFD: out std_logic;
       dct_out: out std_logic;
       ena : inout std_logic;
       control : inout std_logic;
       finaldct : out std_logic;
--       start_dct : out std_logic;
       fin_0 : out std_logic_vector(37 downto 0);
       fin_1 : out std_logic_vector(37 downto 0);
       fin_2 : out std_logic_vector(37 downto 0);
       fin_3 : out std_logic_vector(37 downto 0);
       fin_4 : out std_logic_vector(37 downto 0);
       fin_5 : out std_logic_vector(37 downto 0);
       fin_6 : out std_logic_vector(37 downto 0);
       fin_7 : out std_logic_vector(37 downto 0));
 end;

architecture RTL of control_dct is
type state_type is (datain,processing,transpose,transpose_ready,temp,transpose_delay,idle);
signal state : state_type;
component proj_dct
port(x0,x1,x2,x3,x4,x5,x6,x7: in std_logic_vector(7 downto 0);
    y0,y1,y2,y3,y4,y5,y6,y7: out std_logic_vector(22 downto 0));
end component;

Component transpose_serial
port(data0 : in  std_logic_vector(22 downto 0);
    data1 : in std_logic_vector(22 downto 0);
    data2 : in std_logic_vector(22 downto 0);
    data3 : in std_logic_vector(22 downto 0);
    data4 : in std_logic_vector(22 downto 0);
    data5 : in std_logic_vector(22 downto 0);
    data6 : in std_logic_vector(22 downto 0);
    data7 : in std_logic_vector(22 downto 0);
```

```
    clk   : in std_logic;
    ena   : in std_logic;
  control : in std_logic;
    data8 : out std_logic_vector(22 downto 0);
    data9 : out std_logic_vector(22 downto 0);
    data10 : out std_logic_vector(22 downto 0);
    data11 : out std_logic_vector(22 downto 0);
    data12 : out std_logic_vector(22 downto 0);
    data13 : out std_logic_vector(22 downto 0);
    data14 : out std_logic_vector(22 downto 0);
    data15 : out std_logic_vector(22 downto 0));
end component;

component proj_dct_2
port(x0,x1,x2,x3,x4,x5,x6,x7: in std_logic_vector(22 downto 0);
    y0,y1,y2,y3,y4,y5,y6,y7: out std_logic_vector(37 downto 0));
end component;

--signal lena,lcontrol : std_logic;
signal x0,x1,x2,x3,x4,x5,x6,x7 : std_logic_vector(7 downto 0);
signal Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7 : std_logic_vector(22 downto 0);
signal data8,data9,data10,data11,data12,data13,data14,data15 : std_logic_vector(22 downto
0);
signal twodct_0,twodct_1,twodct_2,twodct_3,twodct_4,twodct_5,twodct_6,twodct_7 :
std_logic_vector(37 downto 0);
begin
chip : proj_dct  port map(X0,X1,X2,X3,X4,X5,X6,X7,Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7);
transpose_k : transpose_serial port
map(data0=>Y0,data1=>Y1,data2=>Y2,data3=>Y3,data4=>Y4,data5=>Y5,data6=>Y6,data7
=>Y7,

clk=>clk,ena=>ena,control=>control,data8=>data8,data9=>data9,data10=>data10,data11=>d
ata11,data12=>data12,data13=>data13,data14=>data14,data15=>data15);
chip_2 :proj_dct_2 port
map(data15,data14,data13,data12,data11,data10,data9,data8,twodct_0,twodct_1,twodct_2,tw
odct_3,twodct_4,twodct_5,twodct_6,twodct_7);


process(clk,rst,ena,control)
variable processing_cnt : integer range 0 to 2;
variable dct_cnt : integer range 0 to 9;
variable some_delay : integer range 0 to 4;
variable count : integer range 0 to 10;
variable rfd_cnt : integer range 0 to 8;
variable cnt_ksk : integer range 0 to 4;
variable next_delay : integer range 0 to 8;
begin
if rst='1' then
fin_0<=(others=>'0');
fin_1<=(others=>'0');
fin_2<=(others=>'0');
```

```
fin_3<=(others=>'0');
fin_4<=(others=>'0');
fin_5<=(others=>'0');
fin_6<=(others=>'0');
fin_7<=(others=>'0');
rfd<='1';
ena<='0';
control<='0';
dct_out<='0';
finaldct<='0';
state<=IDLE;
elsif clk'event and clk='1' then
 if state=datain then
  RFD<='0';
  X0<=DIN_0;
  X1<=DIN_1;
  X2<=DIN_2;
  X3<=DIN_3;
  X4<=DIN_4;
  X5<=DIN_5;
  X6<=DIN_6;
  x7<=DIN_7;
  state<=processing;
 elsif state=processing and processing_cnt<2 then

 processing_cnt:=processing_cnt+1;
 elsif state=processing and processing_cnt=2 then
 Processing_cnt:=0;
 dct_cnt:=dct_cnt+1;
 if dct_cnt < 8 then
 ena<='1';
 control<='1';
 state<=idle;
 elsif dct_cnt=8 then
 dct_cnt:=0;
 ena<='1';
 control<='1';
 rfd<='0';
 state<=transpose;
 else
 null;
 end if;
elsif state=transpose and some_delay < 2 then
ena<='0';
control<='0';
some_delay:=some_delay+1;
state<=transpose;
elsif state=transpose and some_delay=2 then
some_delay:=0;
state<=transpose_ready;
elsif state=transpose_ready and count<10 then
```

```
ena<='1';
control<='0';
if count=1 then
dct_out<='1';
else
dct_out<='0';
end if;
state<=transpose_delay;
elsif state=transpose_delay and cnt_ksk< 4 then
ena<='0';
cnt_ksk:=cnt_ksk+1;
elsif state=transpose_delay and cnt_ksk=4 then
cnt_ksk:=0;
fin_0<=twodct_0;
fin_1<=twodct_1;
fin_2<=twodct_2;
fin_3<=twodct_3;
fin_4<=twodct_4;
fin_5<=twodct_5;
fin_6<=twodct_6;
fin_7<=twodct_7;
count:=count+1;
next_delay:=next_delay+1;
if next_delay<8 then
finaldct<='1';
state<=temp;
elsif next_delay=8 then
next_delay:=0;
finaldct<='1';
state<=temp;
else
null;
end if;
elsif state=temp then
finaldct<='0';
state<=transpose_ready;
elsif state=transpose_ready and count=10 then
ena<='0';
control<='0';
count:=0;
state<=idle;
elsif state=idle and ND='1' then
ena<='0';
dct_out<='0';
if rfd_cnt< 8 then
rfd<='1';
rfd_cnt:=rfd_cnt+1;
elsif rfd_cnt=8 then
rfd_cnt:=0;
rfd<='1';
else
```

```
null;
end if;
control<='0';
state<=datain;
else
null;
end if;
end if;
--lena<=ena;
--lcontrol<=control;
end process;
end;
```

Appendix C

VHDL program for the controller in the DCT using CORDIC algorithm

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use work.array_type.all;

entity cordic_control is
port(data_0 : in signed(16 downto 0);
data_1 : in signed(16 downto 0);
data_2 : in signed(16 downto 0);
data_3 : in signed(16 downto 0);
data_4 : in signed(16 downto 0);
data_5 : in signed(16 downto 0);
data_6 : in signed(16 downto 0);
data_7 : in signed(16 downto 0);
rst : in std_logic;
clk : in std_logic;
start : inout std_logic;
control : inout std_logic;
start_dct : out std_logic;
cordic_out : out std_logic;
second_8 : out signed(16 downto 0);
second_9 : out signed(16 downto 0);
second_10 : out signed(16 downto 0);
second_11 : out signed(16 downto 0);
second_12 : out signed(16 downto 0);
second_13 : out signed(16 downto 0);
second_14 : out signed(16 downto 0);
second_15 : out signed(16 downto 0));
end;

architecture arch of cordic_control is

component one_dimen_dct
port(    clk         : in std_logic;
ena     : in std_logic;
in_data : in in_array;
out_data : out in_array);
end component;

component transpose_now
port(data0 : in signed(16 downto 0);
data1 : in signed(16 downto 0);
data2 : in signed(16 downto 0);
data3 : in signed(16 downto 0);
data4 : in signed(16 downto 0);
data5 : in signed(16 downto 0);
data6 : in signed(16 downto 0);
```

```
data7 : in signed(16 downto 0);
clk   : in std_logic;
ena   : in std_logic;
control : in std_logic;
data8 : out signed(16 downto 0);
data9 : out signed(16 downto 0);
data10 : out signed(16 downto 0);
data11 : out signed(16 downto 0);
data12 : out signed(16 downto 0);
data13 : out signed(16 downto 0);
data14 : out signed(16 downto 0);
data15 : out signed(16 downto 0));
end component;

type state is (idle,one_dct,trans_inter,transpose_ready);
signal st : state;
signal ena : std_logic:='1';
signal
temp_data_0,temp_data_1,temp_data_2,temp_data_3,temp_data_4,temp_data_5,temp_data_
6,temp_data_7 : signed(16 downto 0);
signal temp_0,temp_1,temp_2,temp_3,temp_4,temp_5,temp_6,temp_7 : signed(16 downto
0);
signal dataout_0,dataout_1,dataout_2,dataout_3,dataout_4,dataout_5,dataout_6,dataout_7 :
signed(16 downto 0);
signal tran_0,tran_1,tran_2,tran_3,tran_4,tran_5,tran_6,tran_7 : signed(16 downto 0);
signal second_0,second_1,second_2,second_3,second_4,second_5,second_6,second_7 :
signed(16 downto 0);
signal rasak_0,rasak_1,rasak_2,rasak_3,rasak_4,rasak_5,rasak_6,rasak_7 : signed(16 downto
0);
begin

x1:one_dimen_dct port
map(clk=>clk,ena=>'1',in_data(0)=>temp_data_0,in_data(1)=>temp_data_1,in_data(2)=>tem
p_data_2,in_data(3)=>temp_data_3,
in_data(4)=>temp_data_4,in_data(5)=>temp_data_5,in_data(6)=>temp_data_6,in_data(7)=>t
emp_data_7,out_data(0)=>Temp_0,
out_data(1)=>Temp_1,out_data(2)=>Temp_2,out_data(3)=>Temp_3,out_data(4)=>Temp_4,
out_data(5)=>Temp_5,
out_data(6)=>Temp_6,out_data(7)=>Temp_7);

tr : transpose_now port
map(data0=>dataout_0,data1=>dataout_1,data2=>dataout_2,data3=>dataout_3,data4=>datao
ut_4,
data5=>dataout_5,data6=>dataout_6,data7=>dataout_7,clk=>clk,ena=>start,control=>control
,
data8=>tran_0,data9=>tran_1,data10=>tran_2,data11=>tran_3,data12=>tran_4,data13=>tran
_5,
data14=>tran_6,data15=>tran_7);
```

```
x2 : one_dimen_dct port
map(clk=>clk,ena=>'1',in_data(0)=>second_0,in_data(1)=>second_1,in_data(2)=>second_2,i
n_data(3)=>second_3,
in_data(4)=>second_4,in_data(5)=>second_5,in_data(6)=>second_6,in_data(7)=>second_7,
out_data(0)=>rasak_0,out_data(1)=>rasak_1,out_data(2)=>rasak_2,out_data(3)=>rasak_3,
out_data(4)=>rasak_4,out_data(5)=>rasak_5,out_data(6)=>rasak_6,out_data(7)=>rasak_7);

process(clk,rst)
variable dct_cnt : integer range 0 to 8;
variable block_cnt : integer range 0 to 9;
variable tran_out_cnt : integer range 0 to 9;
variable dct_tran_cnt : integer range 0 to 8;
variable inter_cnt : integer range 0 to 9;
begin
if rst='1' then
start<='0';
control<='0';
start_dct<='0';
cordic_out<='0';
second_8<=conv_signed(0,17);
second_9<=conv_signed(0,17);
second_10<=conv_signed(0,17);
second_11<=conv_signed(0,17);
second_12<=conv_signed(0,17);
second_13<=conv_signed(0,17);
second_14<=conv_signed(0,17);
second_15<=conv_signed(0,17);
st<=idle;
elsif rising_edge(clk) then
if st=one_dct and dct_cnt < 8 then
if dct_cnt<1 then
start_dct<='1';
else
start_dct<='0';
end if;
temp_data_0<=data_0;
temp_data_1<=data_1;
temp_data_2<=data_2;
temp_data_3<=data_3;
temp_data_4<=data_4;
temp_data_5<=data_5;
temp_data_6<=data_6;
temp_data_7<=data_7;
start<='0';
control<='0';
dct_cnt:=dct_cnt+1;
elsif st=one_dct and dct_cnt=8 then
dataout_0<=signed(shr(conv_std_logic_vector(Temp_0,17),"10"));
dataout_1<=signed(shr(conv_std_logic_vector(Temp_1,17),"10"));
dataout_2<=signed(shr(conv_std_logic_vector(Temp_2,17),"10"));
dataout_3<=signed(shr(conv_std_logic_vector(Temp_3,17),"10"));
```

```
dataout_4<=signed(shr(conv_std_logic_vector(Temp_4,17),"10"));
dataout_5<=signed(shr(conv_std_logic_vector(Temp_5,17),"10"));
dataout_6<=signed(shr(conv_std_logic_vector(Temp_6,17),"10"));

dataout_7<=signed(shr(conv_std_logic_vector(Temp_7,17),"10"));
dct_cnt:=0;
start<='1';
control<='1';
block_cnt:=block_cnt+1;
if block_cnt < 8 then
st<=one_dct;
elsif block_cnt=8 then
st<=transpose_ready;
block_cnt:=0;
else
null;
end if;
elsif st=transpose_ready and dct_tran_cnt<8 then
start<='0';
control<='0';
cordic_out<='0';
dct_tran_cnt:=dct_tran_cnt+1;
elsif st=transpose_ready and dct_tran_cnt=8 then
start<='1';
control<='0';
tran_out_cnt:=tran_out_cnt+1;
if tran_out_cnt > 2 then
cordic_out<='1';
second_8<=rasak_0;
second_9<=rasak_1;
second_10<=rasak_2;
second_11<=rasak_3;
second_12<=rasak_4;
second_13<=rasak_5;
second_14<=rasak_6;
second_15<=rasak_7;
end if;
second_0<=tran_7;
second_1<=tran_6;
second_2<=tran_5;
second_3<=tran_4;
second_4<=tran_3;
second_5<=tran_2;
second_6<=tran_1;
second_7<=tran_0;
dct_tran_cnt:=0;
if tran_out_cnt < 9 then
st<=transpose_ready;
elsif tran_out_cnt=9 then
st<=trans_inter;
tran_out_cnt:=0;
```

```
else
null;
end if;
elsif st=trans_inter and inter_cnt < 9 then
control<='0';
cordic_out<='0';
inter_cnt:=inter_cnt+1;
elsif st=trans_inter and inter_cnt=9 then
cordic_out<='1';
second_8<=rasak_0;
second_9<=rasak_1;
second_10<=rasak_2;
second_11<=rasak_3;
second_12<=rasak_4;
second_13<=rasak_5;
second_14<=rasak_6;
second_15<=rasak_7;
st<=idle;
inter_cnt:=0;
elsif st=idle and ena='1' then
cordic_out<='0';
start<='0';
control<='0';
st<=one_dct;
else
null;
end if;
else
null;
end if;
end process;
end;
```