Efficient VLSI Implementation of SVD using CORDIC

A thesis submitted in partial fulfillment of
the requirements for the degree of

Bachelor of Technology

by

Badrinath Singhal and Dewal Agarwal
(Roll No. 140108010 and 140108050)

Under the guidance of
Prof. Shaik Rafi Ahamed

DEPARTMENT OF ELECTRONICS & ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
April 2018

CERTIFICATE

This is to certify that the work contained in this thesis entitled

Efficient VLSI Implementation of SVD using CORDIC

is the work of

Badrinath Singhal and Dewal Agarwal
(Roll No. 140108010 and 140108050)

for the award of the degree of Bachelor of Technology, carried out in the Department of Electronics and Electrical Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.

_____

Guide

Date: _____

Place: _____

DECLARATION

The work contained in this thesis is our own work under the supervision of the guides. We have read and understood the "B. Tech./B. Des. Ordinances and Regulations" of IIT Guwahati and the "FAQ Document on Academic Malpractice and Plagiarism" of EEE Department of IIT Guwahati. To the Best of our knowledge, this thesis is an honest representation of our work.

_____

Author

Date: _____

Place: _____

Acknowledgments

Our deep gratitude goes first to Prof. Shaik Rafi Ahamed sir, who expertly guided us throughout our Bachelor Thesis Project, he was very kind enough to share with us the excitement of his previous works in the related fields. Moreover, he was very understanding towards the problems faced by us during the course of our work.

Our appreciation also extends to our laboratory colleagues. Ajay Kumar Maddirala's mentoring and encouragement have been especially valuable. His early insights and continuous help maintained us on the right course.

Above all it was an unique experience for us to pursue such sort of work, and we are thankful to each and everyone who helped us on the way.

Abstract

In this project we will learn about CORDIC algorithm, its application in SVD computation. We will state SVD algorithm for computation of 2x2 matrix using CORDIC algorithm which can be further implemented in VLSI architecture. We state how to perform matrix multiplication using CORDIC algorithm and state reason on which CORDIC to use out of many available options so that it is easy and efficient to implement. We further state steps which will help to compute SVD of 4x4 matrix using the knowledge of 2x2 matrix. And in the end we end the report by stating conclusion and future work.

Keywords:- CORDIC, Singular Vector Decomposition (SVD), Jacobi algorithm, Off Diagonal Group (ODG), Diagonal Group (DG)

Contents

Abstract **iv**

List of Figures **vii**

1 Introduction **1**

1.1 Electroencephalogram 1

1.2 Singular Spectrum Analysis (SSA) 2

2 CORDIC Algorithm **4**

2.1 Introduction 4

2.2 Basic CORDIC Techniques 5

2.3 Advanced CORDIC Algorithms 8

3 SVD and Matrix Multiplication using CORDIC algorithm **10**

3.1 Singular Value Decomposition/ Eigen Value Decomposition using CORDIC . . 10

3.2 Matrix Multiplication using CORDIC algorithm 11

4 Proposed Approach **13**

4.1 Detailed Algorithm 13

4.2 Strategy for parameter computations 14

4.2.1 ϕ computation strategy 14

4.2.2 Strategy for direct rotation 15

4.2.3 Diagonal group computation 15

4.2.4 Non diagonal group computation 16

4.2.5 Algorithm for Interchange 16

5 Results, Conclusion and future Work **18**

5.1 Time Complexity 18

5.2 Results 18

5.3 Conclusion 22

5.4 Future Work 22

List of Figures

2.1 Vector Rotation 6
2.2 CORDIC Block Diagram (ref. from [2]) 7

4.1 Schematic for flow of information 17

5.1 Variation of Average Error w.r.t the Number of Iterations 19
5.2 Variation of Average Error w.r.t the Range of Random Numbers in Matrix . . . 20
5.3 CORDIC Simulation 45 degree(Verilog) 21
5.4 CORDIC Simulation 90 degree (Verilog) 21

Chapter 1

Introduction

1.1 Electroencephalogram

Electroencephalogram (EEG) signals records electrical patterns generated in brain. Nerve cells in general produces electric signals, presence of billions of nerve cells in brain together form a pattern called brain waves. During EEG small electrodes and wires are attached to head of a subject and that electrodes detects brain waves generating continuously. EEG then amplifies and record the signal. EEG signals are used to measure seizure, epilepsy, head injuries, tumours etc.

While extracting desired information through electroencephalogram (EEG) signals it is necessary to apply signal processing techniques to deal with the artifacts present in them. EEG signals are generally mixed with Electromyography (EMG) and Electrooculography (EOG) signals. EMG signals are electrical activity produced by skeletal muscles whereas EOG signals is recording of corneo-retinal standing potential that exist between front and back of human eye. The performance of various algorithm like seizers detection, BCI and other features degrade due to presence of these artifacts. So the removal of these artifacts becomes important step in analysis of EEG signals. In practice removal of EMG and EOG artifacts are dealt separately for multi channel EEG signals. This is because of the fact that both EMG and EOG signals posses different properties and cannot be treated as same.

The utilization of low pass filter are utilized to expel the EMG artifacts EEG signals. Despite

the fact that the utilization of low pass channel may change the EEG signal, this is due to the fact that EMG and EEG signals have overlapping spectrum. Another method named Independent Component Analysis (ICA) is generally used for removing EMG artifacts.

EOG artifacts are usually handled by utilizing the technique of noise cancellation. Noise canceller is usually used to get rid of artifacts from bio-medical signals. It is generally used with ICA to detect independent components representing EOG artifacts. However, this method is used for multichannel EEG signals and cannot be used to in case of portable devices.

1.2 Singular Spectrum Analysis (SSA)

Singular spectrum analysis technique is proposed to isolate EMG and EOG artifacts from singular channel EEG signals. Firstly the single channel signal is mapped to multi channel signals (S) by shifting the signal by one and embedding it.

$$\mathbf{S} = \begin{pmatrix} s(1) & s(2) & \dots & \dots & s(k) \\ s(2) & s(3) & \dots & \dots & s(k+1) \\ \vdots & \vdots & \dots & \dots & \vdots \\ s(M) & s(M+1) & \dots & \dots & s(N) \end{pmatrix}.$$

SVD is used to figure orthogonal eigenvectors, here V and U are identity matrices and D is diagonal matrix. The co-variance C for S represent the corresponding eigenvalues and eigenvectors as $\lambda_1, \lambda_2, \dots, \lambda_M$ and $v_1, v_2, \dots, v_M$ respectievly. The eigenvectors and corresponding eigenvectors expressed are in descending order.

$$u_i = \frac{S^T v_i}{\sqrt{\lambda_i}}$$

where i= 1, 2, 3 . . . , M. And the trajectory matrix can be defined as in

$$S = \sum_{i=1}^{M} S_i = \sum_{i=1}^{M} \sqrt{\lambda_i} v_i u_i^T$$

Later we tabulate variations of each eigenvector.

$$m_y = \frac{\sqrt{\frac{\sum_{j=1}^{N} y(j)^2}{N}}}{\sqrt{\frac{\sum_{j=1}^{N-1} d(j)^2}{N-1}}}$$

where m_y is local variations of signal y = [y(1), y(2),. . . ,y(N)] and *d(j)= y(j) - y(j-1).*

The multichannel signal is then projected after setting threshold according to need in the domain covered by eigenvectors whose variations are less than mentioned threshold.

Chapter 2

CORDIC Algorithm

2.1 Introduction

The popularity of CORDIC has since then enhanced because of its potential for productive and minimal effort executions of an assortment of uses. CORDIC has turned out to be favorable over different calculation strategies because of straightforwardness of its hardware usage, as it allows the use basic shift-add operations of the forms:

$$a + b.2^{-i}$$

$$a - b.2^{-i}$$

The second part of the chapter discusses the principles of CORDIC operations, covering the elementary ideas form coordinate transformation to rotation mode and vectoring mode operations followed by basic multidimensional CORDIC.

In order to implement CORDIC algorithms it is very important to understand the key developments that have taken place in implementation of CORDIC algorithms and CORDIC based architectures. So the third part of this chapter, deals with various types of CORDIC algorithms.

Later in the chapter we have discussed the standard applications of CORDIC algorithm, the ones that are relevant to the development of project. It also mentions the reasons behind the

choice made.

2.2 Basic CORDIC Techniques

The rotation of a vector in two-dimensional space T_k through an angle ϕ can obtained using following iterative steps. We define a rotational matrix as R, for an angle of rotation ϕ in the coordinate plane as:

$$\mathbf{R} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} 5$$

The final vector after undergoing rotation can be represented as T_{n+1}. This vector can be obtained from its initial value after a series of steps. However we can summarize the entire operation as below:

$$\mathbf{T_{k+1}} = \begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x_k \\ y_k \end{bmatrix}$$

The above rotation matrix can be factored using cosine term, and rewritten as

$$\mathbf{R} = \left[(1+\tan^2\phi)^{-1/2}\right] \begin{bmatrix} 1 & -\tan\phi \\ \tan\phi & 1 \end{bmatrix}$$

and can be interpreted as a product of scalar-factor K = $[(1+\tan^2\phi)^{-1/2}]$ and a modified rotation matrix R_m given as:

$$\mathbf{R_m} = \begin{bmatrix} 1 & -\tan\phi \\ \tan\phi & 1 \end{bmatrix}$$

However this results in the change in the value of the initial value final vector T_{k+1} by a scaling factor $K = \cos\phi$. Thus lets call the new scaled vector as $T'_{k+1} = R_m T_k$. To achieve this we follow following set of steps (i) The rotation angle ϕ is broken into elementary rotations through predefined angle, and (ii) Avoid the scaling factor as it may cause additional computational costs for square root calculation.

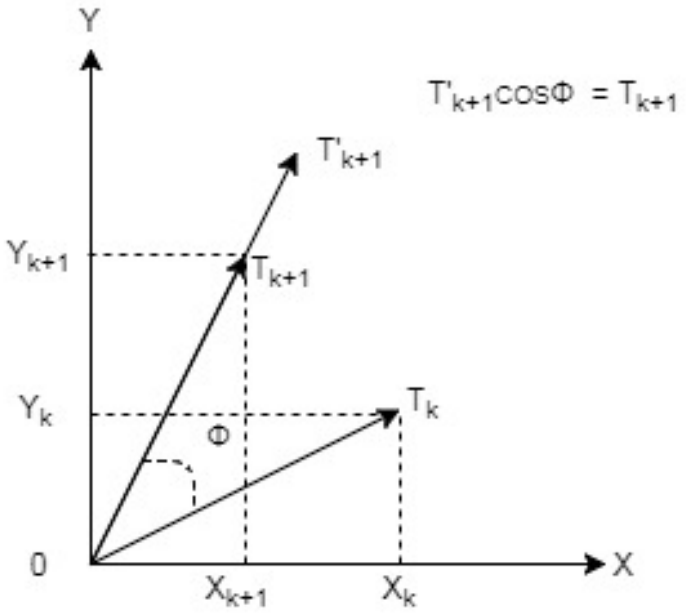


Figure 2.1: Vector Rotation

1) Decomposing the Angle of Rotations: The CORDIC algorithm performs the rotation iteratively by breaking down the angle of rotation into smaller predefined angles, β_i = arctan(2^{-i}), Thus β_i can be written in the form micro-rotations as:

$$\phi = \sum_{i=0}^{i=n-1} \sigma_i \beta_i \text{ and } \sigma_i = 1 \text{ or } -1.$$

that satisfies the CORDIC convergence theorem :- β_i - $\sum_{j=i+1}^{i=n-1} \sigma_i < \beta_{n-1}$, $\forall$ i, i = 0,1,....,n-2. The mentioned decomposition of ϕ can only be used for -1.7432 $\leq \phi \leq$ 1.7432. Keeping this in mind we can define Ω_0 = 0 and Ω_{i+1} = Ω_i - $\sigma_i.\beta_i$ with σ_i = 1 if $\Omega_i \geq 0$ and σ_i = -1 otherwise.

Thus the rotation matrix R for the ith iteration is given as-

$$\mathbf{R(i)} = K_i \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix}$$

K_i = $\frac{1}{\sqrt{(1+2^{-2i})}}$ is the scaling factor, and modified rotaion matrix

$$\mathbf{R_m(i)} = \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix}$$

Here K_i is independent of σ_i.

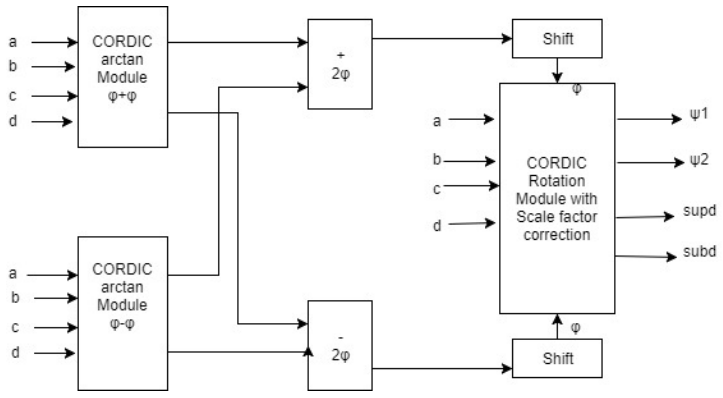


Figure 2.2: CORDIC Block Diagram (ref. from [2])

2) Avoiding Scaling: For simplification the scale-factor $K_i = \frac{1}{\sqrt{(1+2^{-2i})}}$ is removed from the rotation matrix expression $R(i)$. The removal lead to $T'_{k+1} = R_m T_k$ instead of desired $T_{k+1} = KR_m T_k$, where K is

$$K = \prod_{i=0}^{n} K_i = \prod_{i=0}^{n} \frac{1}{\sqrt{(1+2^{-2i})}}$$

The basic CORDIC iteration are obtained by applying the modified rotation of the vector $T'_{i+1} = R_m(i)T_i$ as follows:

$$x_{i+1} = x_i - \sigma_i 2^{-i}.y_i$$

$$y_{i+1} = y_i + \sigma_i 2^{-i}.x_i$$

$$\Omega_{i+1} = \Omega_i + \sigma_i.\beta_i$$

The above set of equation can be used for both modes of operation. To overcome the problem of limited convergence we extend CORDIC rotations to complete range of +π or -π we initialize the equation as, through an initial rotation +π/2 or -π/2 -

$$x_0 = -\sigma_{-i}.y_{-i}$$

$$y_0 = +\sigma_{-i}.x_{-i}$$

$$\Omega_0 = \Omega_{-i} + \sigma_{-i}.\beta_{-i} \qquad \text{where } \beta_{-i} = \pi/2.$$

2.3 Advanced CORDIC Algorithms

This section deals with the standard CORDIC Algorithms that is an extension of the basic CORDIC algorithm discussed in the previous sections. The two basic features of the basic CORDIC Algorithm are: a) The rotation for iterations is performed on the intermediate vector computed from the previous iterations. b) The $(i + 1)_t h$ iteration requires the completion of previous iteration.

The advanced CORDIC Algorithms vary from each other on the basis of the fact that how are they able to handle the above listed bottle necks. Thus, their performance upon architectural implementation depend largely upon these two facts. The following have been chosen for further implementation purposes-

(I) *Angle Recoding (AR) Methods*

(II) *Differential CORDIC Algorithm*

(I) ***Angle Recoding (AR) Methods***: As the number if iterations play a very crucial role in the performance of CORDIC algorithm, thus it becomes very important that we incorporate such a strategy that enables us to decrease the number of recursive steps. In traditional CORDIC the angle ϕ was expressed as the linear combination of elementary angles β, the values belonged to the set $P = \{(\sigma.\tan^{-1}(2^{-i}))\} : \sigma \in \{-1, 1\}, i \in \{1, 2, 3..., n-1\}$ in $\phi = \sum_{i=0}^{i=n-1}[\sigma_i \tan^{-1} 2^{-}i]$.

Now to reduce the number of iterations zeros are added to the linear combination to get the desired angle ϕ in relatively fewer iterations for $P = \{(\sigma.\tan^{-1}(2^{-i}))\} : \sigma \in \{-1, 0, 1\}, i \in \{1, 2, 3..., n-1\}$. %.

(II) ***Differential CORDIC Algorithm***: This algorithm is equivalent to the basic CORDIC form with respect to the accuracy and convergence. However it uses redundant number system and introduces some temporary variables corresponding to CORDIC variables x, y and Ω and is defined as-

$$\hat{\rho_{i+1}} = sign(\rho_i).\rho_{i+1}$$

which implies $|\hat{\rho_{i+1}}| = |\rho_{i+1}|$ and sign(ρ_{i+1}) = sign(ρ_i).sign($\hat{\rho_{i+1}}$). Thus the signs of ρ_{i+1} are encoded in $\hat{\rho_{i+1}}$.

For rotation mode we have following set of equations-

$$x_{i+1} = x_i - sign(\Omega_i).2^{-i}.y_i$$

$$y_{i+1} = y_i + sign(\Omega_i).2^{-i}.x_i$$

$$|\hat{\Omega_{i+1}}| = ||\hat{\Omega}_i| - \beta_i|$$

$$sign(\Omega_{i+1}) = sign(\Omega_i).sign(\hat{\Omega_{i+1}})$$

Chapter 3

SVD and Matrix Multiplication using CORDIC algorithm

CORDIC is used for some basic matrix problems like QR decomposition and singular-value decomposition. CORDIC is also applicable to signal and image processing, digital communication, robotics etc. CORDIC have found much success in matrix problems because of its ease of implementation and using adder and shifter to implement complex computations. We will discuss in further section about implementation of CORDIC in computing SVD and Matrix Multiplication.

3.1 Singular Value Decomposition/ Eigen Value Decomposition using CORDIC

Singular Value Decomposition of a matrix M is given by $M = U\Sigma V^T$, where U and V are orthogonal matrices and Σ is diagonal matrix with values of square of eigen values sorted in descending order.

SVD and EVD have a wide range of applications in image processing, signal processing, robotics and communication. So implementation of SVD and EVD efficiently becomes an important task. Using CORDIC, SVD and EVD estimation which involves multiplication, divi-

sion, square root by CORDIC modules which is almost twice as fast.. In this sections we state the use of CORDIC in computing SVD and VLSI architecture for the same.

Let given matrix be S, so SVD of **S** is $S = UDV^T$. The dimension of S be pxp, To start with CORDIC we have to distribute pxp matrix S to an array of $\frac{p}{2}$x$\frac{p}{2}$ simple 2x2 matrix. 2x2 SVD is defined as

$$R(\phi_l)^T \begin{bmatrix} a & b \\ c & d \end{bmatrix} R(\phi_r) = \begin{bmatrix} \psi & 0 \\ 0 & \psi \end{bmatrix}$$

where ϕ_l and ϕ_r are left and right rotation angles respectively. The rotation matrix is

$$\mathbf{R}(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$$

We can calculate ϕ_l and ϕ_r by simplifying above equation to get

$$\phi_{SUM} = \phi_l + \phi_r = \tan^{-1}\frac{c+b}{d-a}$$

$$\phi_{DIFF} = \phi_r - \phi_l = \tan^{-1}\frac{c-b}{d+a}$$

We calculate D using the above equations for every 2x2 matrix present in an array. Later we embed each of D matrix to make pxp matrix. At last we apply transform the resultant matrix to the diagonal matrix which gives us the required eigenvalue matrix and U and V. We use Angle Recoding CORDIC to make it more efficient as Angle Recoding CORDIC works 50% faster than basic CORDIC stated in chapter 2.

3.2 Matrix Multiplication using CORDIC algorithm

Other important part of this project is to perform Matrix Multiplication using VLSI architecture. The goal is to implement Matrix Multiplication using minimum amount of extensive operations like multiplications, divisions etc which can make the process slow and increase the throughput. Let us take the example of Matrix Multiplication below.

$$\begin{bmatrix} a_1 & a_2 & \dots & a_M \\ a_{M+1} & a_{M+2} & \dots & a_{2M} \\ \vdots & \vdots & \dots & \vdots \\ a_{MN-M} & a_{MN-M+1} & \dots & a_{MN} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix}$$

The $1st$ row of the resultant matrix will be

$$a_1x_1 + a_2x_2 + \ldots + a_Mx_M$$

express each of a_i as sum of the powers of 2's. Let a_1 be 13 then a_1= 2^3 + 2^2+2^0. That means a_1x_1 will be $x << 3$+$x << 2$+$x << 0$, here $x << i$ denotes bit wise right shift of x by i.

We do this shift and addition for each term of the expression and we add the resultant of each term to obtain the value. This method saves us from expensive multiplication process, it changes all the multiplication to shifting and addition process. There are other more efficient computations are available which can do the similar task

Chapter 4

Proposed Approach

In previous chapter we have discussed about algorithm for SVD computation for 2x2 matrix, in this chapter we will discuss about SVD computation algorithm for 4x4 matrix in detail. We will discuss about implementation methodology and given an example of our algorithm by assuming a 4x4 matrix and solving it step by step using our proposed algorithm.

We want to scale SVD computation using SVD processor to reduce the computation time. There are large number of applications where this can be successfully implemented and could potentially save us lot of computational resources. Traditionally we have been approaching this problem by breaking mxm matrix in $\frac{m}{2}$x$\frac{m}{2}$ matrices of 2x2 and later clubbing them such that it results SVD of the matrix. Its obvious that there is a need of a much better approach for dealing with this problem. And in this chapter we will explore the same.

4.1 Detailed Algorithm

There are plenty of methods which can be used to calculate SVD. Jacobi algorithm is generally preferred because of its simplicity, parallelism and easy to implement. At first we will reduce our Singular Value Decomposition problem to computation of symmetric Eigen Value Decomposition problem. Two sided Jacobi rotation for SVD computations, the matrix is diagonalised by following

$$A = J(g, h, \phi)^T * A * J(g, h, \phi)$$

for all (g,h) combinations, g,h ϵ 1 to N..

We can see from the equation that multiplying by left rotation matrix modifies $g^(th)$ and $h^(th)$ rows, multiplying by right rotation matrix modifies $g^(th)$ and $h^(th)$ column. We will use this statement to compute SVD of 4x4 by carefully selecting elements to be manipulated. Lets consider a general 4x4 matrix whose SVD is to be computed through CORDIC.

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

We will deal separately deal with each group of 4 elements of 4x4 matrix. We are separating the groups in 2 groups i.e. diagonal group (DG) and off diagonal group (ODG). 1st diagonal group consists of elements $a_{11}, a_{12}, a_{21}, a_{22}$ and 2nd diagonal group of $a_{33}, a_{34}, a_{43}, a_{44}$. Similarly 1st off diagonal group will consist of elements $a_{13}, a_{14}, a_{23}, a_{24}$ and second group of $a_{31}, a_{32}, a_{41}, a_{42}$, the same can be seen in below figure.

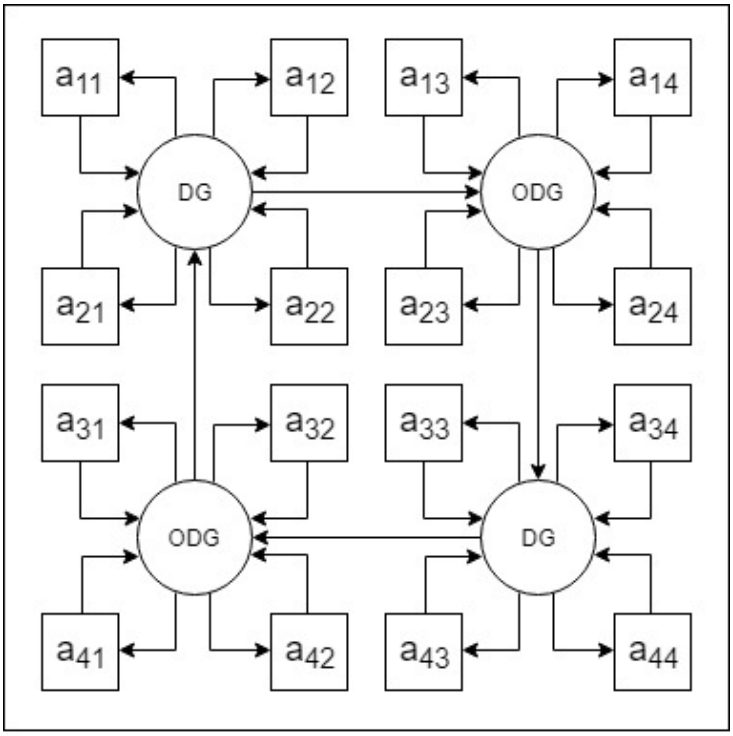
The reason we divided the matrix in 2 groups because each group will perform function different than the other. The diagonal group will be used to calculate ϕ and propagate the rotation parameters to rows (g,h) and columns. The off diagonal group will get the rotation parameters from diagonal group and rotate the 2x2 matrix. When left iteration is done values are stored back as it is but after right rotation the values are interchanged and then stored.

In next section we will discuss about strategies for ϕ computation, direct rotation and diagonal processing which will be used to achieve Eigen Value decomposition with minimum computation.

4.2 Strategy for parameter computations

4.2.1 ϕ computation strategy

Since the aim is to implement the SVD computation to architecture so the strategy should be applicable for the same. Here diagonal group calculate ϕ using CORDIC algorithm. CORDIC



DG : DIAGNOL ELEMENT
ODG : OFF DIAGNOL ELEMENT

Systolic Architecture for 4X4

will be implemented to compute $\tan^{-1}\frac{y}{x}$. Here y will be 2*b and x will be d-a. This phi is them propagated to (g,h) columns and rotation is applied to the matrix using CORDIC.

4.2.2 Strategy for direct rotation

During CORDIC rotation mode we are generating direction of rotation again from ϕ. We use different approach than computing ϕ, propagating ϕ and then again evaluating direction of rotation, we directly send direction to off diagonal group to rotate the matrix. This have doesn't have much difference for implementing it in software but while developing its VLSI architecture it does not need extra register for storing ϕ. This helps in easy computation as the size of the matrix increases.

4.2.3 Diagonal group computation

Diagonal group is used to diagonalise the matrix. During the same, we propagate the direction of rotation to ODG. During left rotation, the matrix is diagonalised and the direction of rotation

is send to ODG. Without storing we apply same operation for right rotation and direction is send to columns. Later we interchange the diagonalised matrix and store it. We diagonalise the matrix through CORDIC.

$$\begin{bmatrix} cos(\phi) & sin(\phi) \\ -sin(\phi) & cos(\phi) \end{bmatrix} = cos(\phi) \begin{bmatrix} 1 & tan(\phi) \\ -tan(\phi) & 1 \end{bmatrix}$$

Here s=tan(ϕ) is implemented using CORDIC. We repeat this step in CORDIC for s = 2^{-i}, for i=1 to 15.

Diagonal group have 4 inputs, four numbers for 2x2 matrix. The CORDIC rotation is performed for i=1 to 15.

4.2.4 Non diagonal group computation

Non diagonal group have 7 inputs i.e the four elements of the matrix, row, direction i and direction j. The output is 4 elements. This group just perform rotation of the matrix given the rotation angle. The row input indicates whether row operations is to be performed or column operation.

4.2.5 Algorithm for Interchange

In this subsection we will discuss about the algorithm for data interchange between diagonal and off diagonal groups. So far now all the four groups were working independent of each other i.e. operation performed on one group doesn't depend on other group. Below we have discussed the way to perform data interchange between groups so that matrix is connected and SVD is evaluated.

The data interchange performed by processor UNIT of the hardware schematic as follows:

$$\text{if i=1 and j=1 then } \begin{bmatrix} outa \leftarrow a & outb \leftarrow b \\ outc \leftarrow c & outd \leftarrow d \end{bmatrix}$$

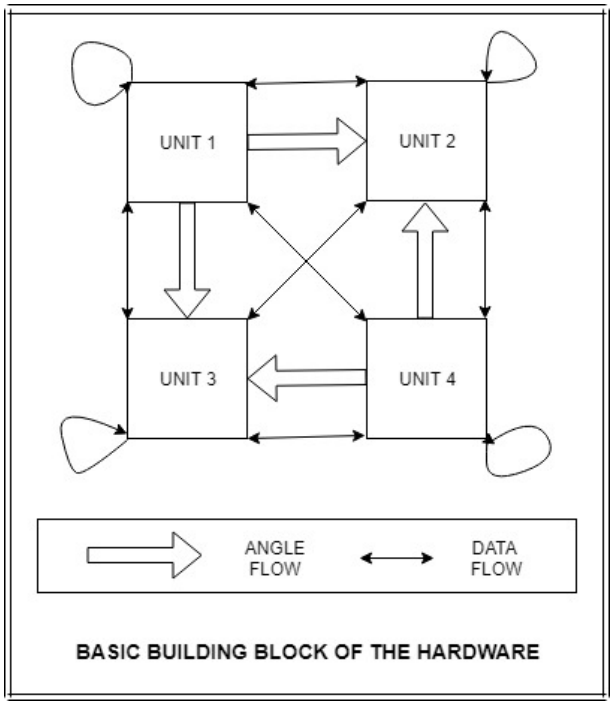


Figure 4.1: Schematic for flow of information

$$\text{else if i=1 then } \begin{bmatrix} outa \leftarrow b & outb \leftarrow a \\ outc \leftarrow d & outd \leftarrow c \end{bmatrix}$$

$$\text{else if j=1 then } \begin{bmatrix} outa \leftarrow c & outb \leftarrow d \\ outc \leftarrow a & outd \leftarrow b \end{bmatrix}$$

$$\text{else } \begin{bmatrix} outa \leftarrow d & outb \leftarrow c \\ outc \leftarrow b & outd \leftarrow a \end{bmatrix}$$

The above algorithm is applied for every UNIT and that shows what data flows out of each output pin of a group. And the input pin of group stores the data at same place it receives. The output pins of the UNIT are connected to each other as shown in above figure.

After one full update of whole matrix we then perform the same operation again i.e. diagonalising DG and rotating ODG.

In this chapter we have discussed the strategy for computation of SVD for 4x4 matrix in detail. We have perform some simulations for the same and compared output with actual SVD of the same matrix which we discuss in next chapter.

Chapter 5

Results, Conclusion and future Work

5.1 Time Complexity

In this algorithm we computed SVD of a matrix using cyclic Jacobi method. For a matrix of size mxm the algorithm requires $O((m^2))$ processors. The algorithm takes $O(mlog(m))$ units of time on an average to complete the operation.

5.2 Results

In this section we discuss about results we acquired from the simulations performed. We compared the evaluated eigenvalues to the actual eigenvalues and plotted the error w.r.t. number of iterations. We can see that as we increase number of iterations the accuracy of the algorithm increases. We have tested the algorithm with over 20 matrices for each iteration as well as for matrices containing entries between 0 to 1000, then we averaged them to get error. We also tested our algorithm with large value of elements in the matrix and plotted them to see the general trend.

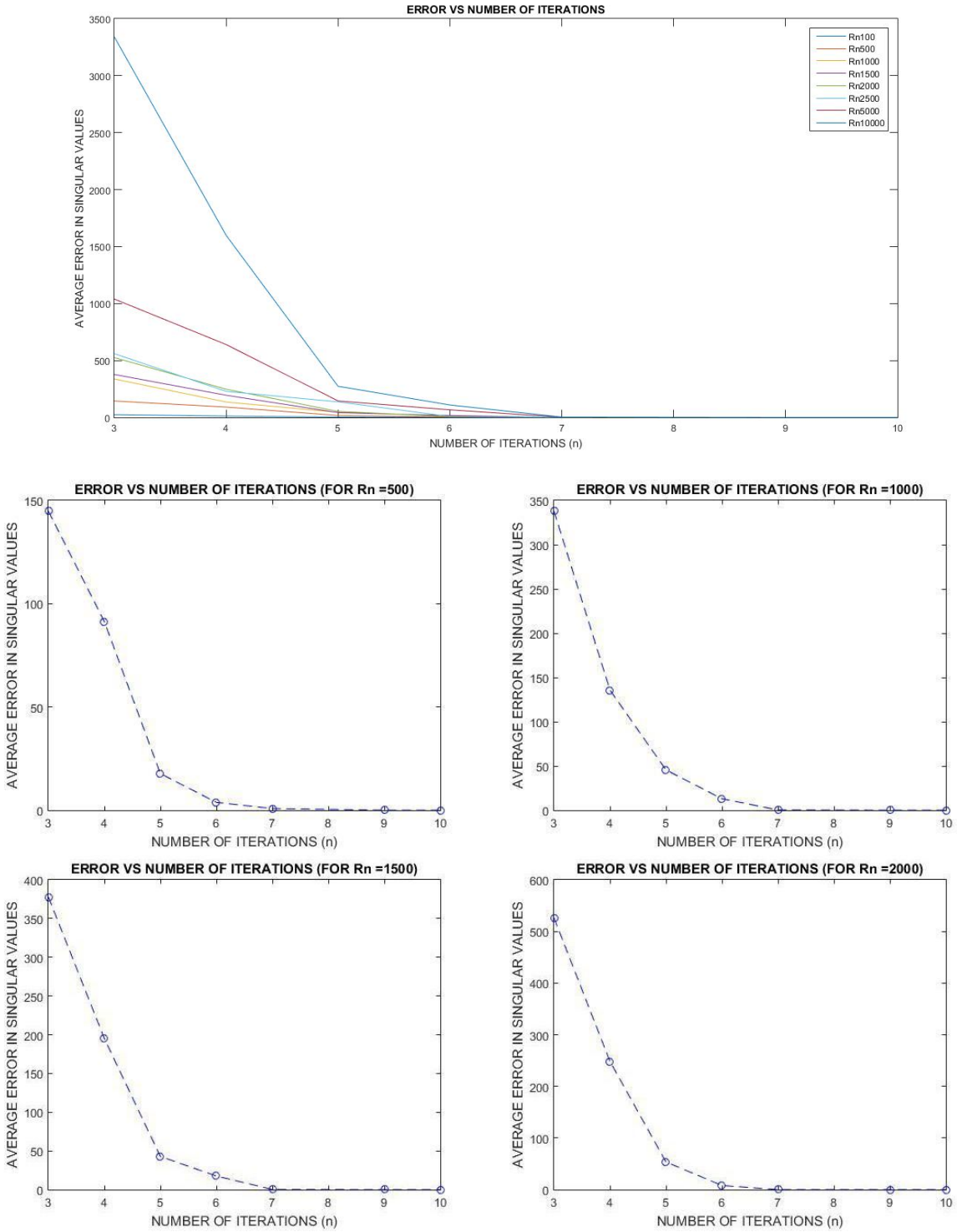


Figure 5.1: Variation of Average Error w.r.t the Number of Iterations

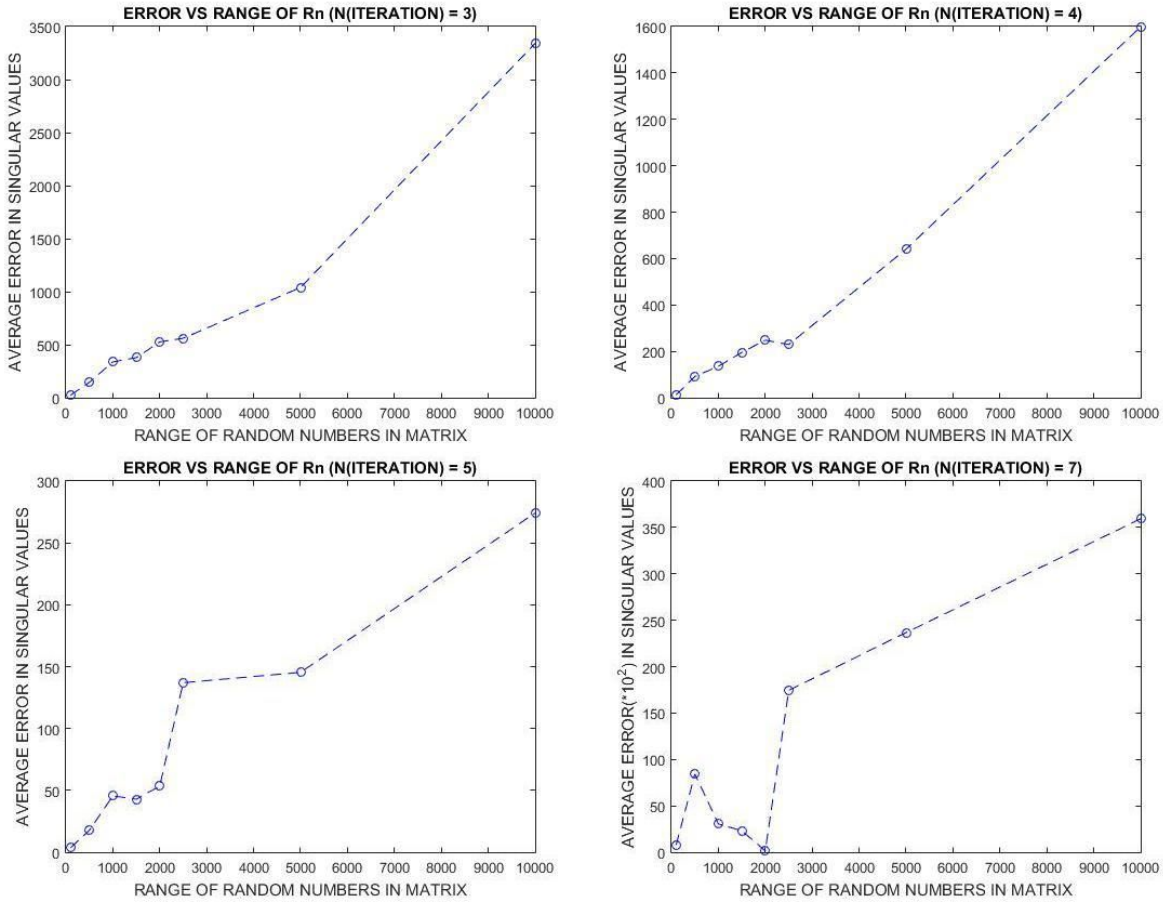


Figure 5.2: Variation of Average Error w.r.t the Range of Random Numbers in Matrix

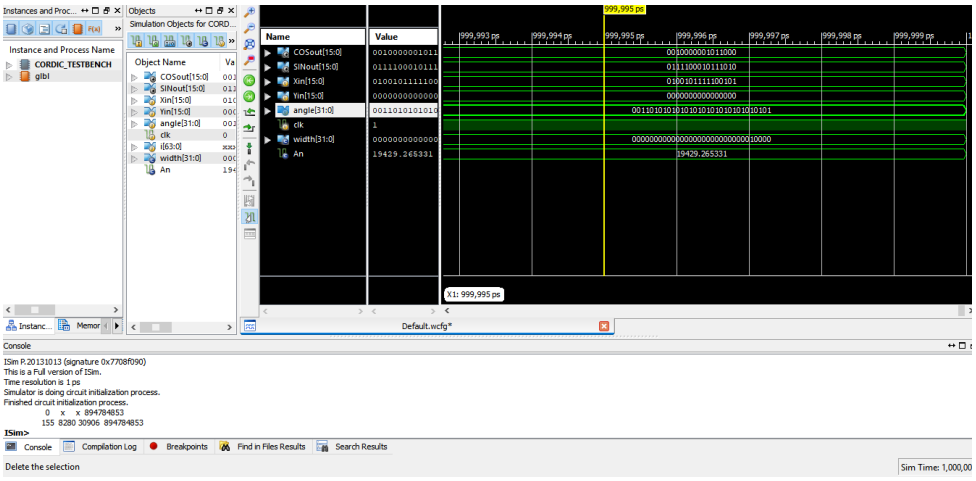


Figure 5.3: CORDIC Simulation 45 degree(Verilog)

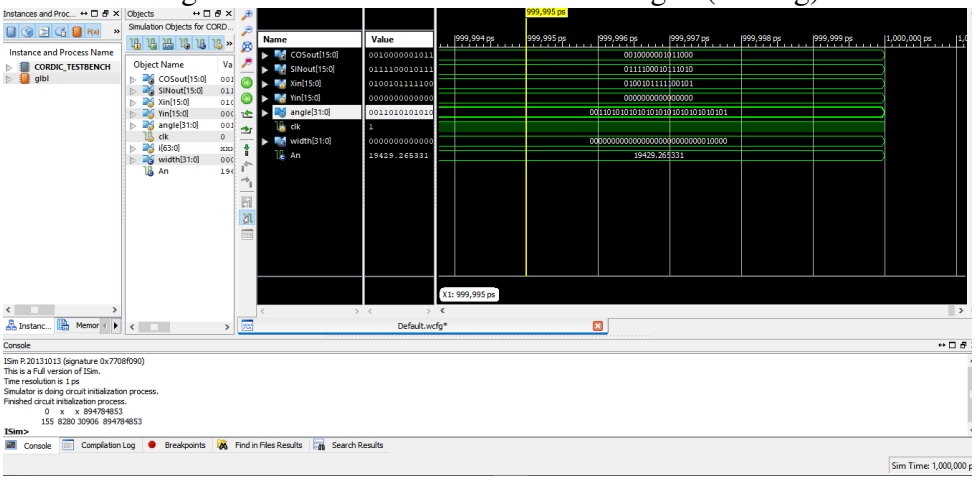


Figure 5.4: CORDIC Simulation 90 degree (Verilog)

5.3 Conclusion

In this project we read about CORDIC, Singular Value Decomposition, Eigenvalue decomposition and how to implement that in a VLSI architecture. We figured SVD of 2x2 matrix using CORDIC, we showed a iterative method to achieve CORDIC rotation by splitting the angle into the inverse tangent of the negative power of 2's. To calculate SVD of matrix of size higher than that its been proposed that we break the matrix into many 2x2 smaller matrix and we apply our SVD approach to that. This lead to huge computational complexity as size of matrix increases.

In this project we formally stated algorithm to compute SVD of 4x4 matrix which doesn't require it to split it in 4 2x2 matrix. We stated how the parameters are transferred to adjacent group and how to perform operation to that group.

5.4 Future Work

This project can be move forward by implementing the 4x4 SVD algorithm in verilog . It is also possible to expand the project by figuring method for SVD of higher dimensional matrix i.e. mxm where m greater than 4. We can use this algorithm to remove artifact from single channel EEG signal.

Bibliography

[1] Karthikeyan N, Sathyanarayan S Vamsi Krishna S and Shankar Balachandran"FPGA implementation of Singular Value Decomposition".

[2] Meher, Pramod Kumar; Valls, Javier; Juang, Tso-Bing; Sridharan, K.; Maharatna, Koushik"50 Years of CORDIC: Algorithms, Architectures and Applications," *IEEE Transactions on Circuits and Systems-I: Regular Papers (published 2009-09-09). 56 (9): 18931907.*, 2016.

[3] 'H. Dawid and H. Meyr,'An overview of parallel algorithms for the singular value and symmetric eigenvalue problems" *Journal of Computational and Applied Mathematics*, Vol. No. 27, pp. 191-213, Sept 1989

[4] 'H. Dawid and H. Meyr,'The differential CORDIC algorithm: constant scale factor redundant implementation without correcting iterations," *IEEE Transactions on Computers,*, Vol. 8, No. 3, pp. 330-334.

[5] J. R. Cavallaro and F. T. Luk, 'CORDIC arithmetic for an SVD processor," *Journal of Parallel and Distributed Computing,*, Vol. 5, No. 3, pp. 271-290 1988

[6] Rafi Ahamed Shaik and Ajay Kumar Maddirala, "Separation of artifacts from electroencephalogram signal using sequential singular spectrum analysis." *2015 International Conference on Signal Processing and Communication Engineering Systems*.

[7] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Transactions on Electronic Computers*, Vol. 8, No. 3, pp. 330-334 1959.

[8] Marc Moonen, Paul Van Dooren and Joos Vandewalle,'A systolic array for SVD updating," *SIAM Journal on Matrix Analysis,*, Vol No. 13, pp. 1015-31038

[9] 'H. Dawid and H. Meyr,'An overview of parallel algorithms for the singular value and symmetric eigenvalue problems" *Journal of Computational and Applied Mathematics*, Vol. No. 27, pp. 191-213, Sept 1989

[10] Ajay Kumar Maddirala; Rafi Ahamed Shaik, "Removal of EMG artifacts from single channel EEG signal using singular spectrum analysis" *2015 IEEE International Circuits and Systems Symposium (ICSyS)*.

[11] J. S. Walther,"A unified algorithm for elementary functions" *Proc. 38th Spring Joint Computer Conf., Atlantic City, NJ, 1971,* Vol.EC-8, No. 3, pp. 330-334 Sept 1959