An Efficient FPGA Implementation of Singular Value Decomposition

A project report submitted in partial fulfilment of the requirements for the degree of B.Tech Electronics and Communication Engineering with specialization in Design and Manufacturing

by

Namala Vindya Vahini
(Roll No: 121EC0036)

Under the Guidance of
Dr. Mohamed Asan Basiri M
Assistant Professor
Dept. of ECE, IIITDM Kurnool

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
INDIAN INSTITUTE OF INFORMATION TECHNOLOGY DESIGN AND MANUFACTURING KURNOOL

April 2025

Evaluation Sheet

Title of the Project: An Efficient FPGA Implementation of Singular Value Decomposition

Name of the Student: Namala Vindya Vahini

Examiner(s):

Supervisor(s):

Head of the Department:

Date:

Place:

Certificate

I, **Namala Vindya Vahini**, with Roll No: **121EC0036** hereby declare that the material presented in the Project Report titled **An Efficient FPGA Implementation of Singular Value Decomposition** represents original work carried out by me in the **Department of Electronics and Communication Engineering** at the **Indian Institute of Information Technology Design and Manufacturing Kurnool** during the years **2024 - 2025**. With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Namala Vindya Vahini (121EC0036)

Date: Student's Signature

In my capacity as supervisor of the above-mentioned work, I certify that the work presented in this Report is carried out under my supervision, and is worthy of consideration for the requirements of B.Tech. Project work.

Advisor's Name: **Dr. Mohamed Asan Basiri M** Advisor's Signature

Abstract

Singular value decomposition, which is a crucial concept in linear algebra and data analysis, finds its application in various fields that include dimensionality reduction, noise reduction, learning machines, and image compression. This initiative emphasizes on devising optimal implementations of SVD with better performance and accuracy. A careful evaluation of both historical and modern methods, such as Jacobi SVD, Hestenes SVD and randomized SVD, is performed, in terms of their computational efficiency and scalability. The aim is to show how computations can be simplified without merciless precision so that SVD may be applied in large-scale data analytics and problems of high-dimensional data. In doing so, the present research argues by practice, through case studies, benchmark tests and practical tasks how the new SVD implementations can transform data processing pipelines and improve the efficiency of the entire system.

Acknowledgements

I express my deepest appreciation to my mentor **Dr. Mohamed Asan Basiri M**, for his unwavering guidance and support throughout the project. His vast knowledge, valuable advice, and patient guidance have been invaluable in every stage of our research. His experience and professional expertise have played a crucial role in successful completion of the research.

I would also like to extend my appreciation to my friends and family for their unwavering support and encouragement. Their constant encouragement and support have been the driving force behind my success.

Finally, I would like to thank you all the resources and references, including books, research papers, online resources, and software, that have helped me.

Contents

List of Figures

List of Tables

Abbreviations

SVD	**S**ingular **V**alue **D**ecomposition
VLSI	**V**ery **L**arge **S**cale **I**ntegration

Chapter 1

Introduction

1.1 Understanding Singular Value Decomposition (SVD)

SVD or singular value decomposition [4] is one of the principal techniques in Linear Algebra where a given matrix can be partitioned into multipliers, in this case, singular values. There are immense computational purposes of SVD in data science, machine learning, image processing, and compression, and also in signal processing. Furthermore, SVD aids in the removal of noise, abbreviating the set of features needed, and denoting the feature construction processes.

The SVD of a matrix A of dimension $m \times n$ can be expressed as:

$$A_{m\times n} = U_{m\times m} \cdot \sigma_{m\times n} \cdot V^{T}_{n\times n} = U_{m\times m} \cdot \sigma_{m\times n} \cdot V^{-1}_{n\times n}$$

For example:

$$a = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

In this decomposition:

- U is an orthogonal matrix whose columns are the left singular vectors.

- σ is a diagonal matrix with non-negative real numbers called singular values.
- V^T is the transpose of an orthogonal matrix V whose columns are the right singular vectors.

The singular values are ordered as $\sigma_1 \geq \sigma_2 \geq 0$. In σ, non-diagonal elements are zero, where σ_1 and σ_2 represent scaling along orthogonal axes. A value of 1 indicates no scaling, while values greater than or less than 1 indicate stretching or compression, respectively.

The geometric interpretation of SVD can be viewed as a sequence of linear transformations:

$$\text{Transform} = \text{Rotation}(V^T) \rightarrow \text{Scaling}(\sigma) \rightarrow \text{Rotation}(U)$$

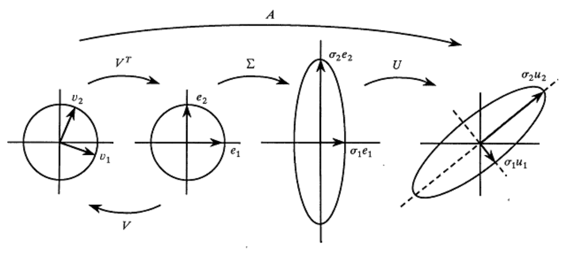


FIGURE 1.1: Example of Rotation and Stretching in SVD [1]

The following identities arise from SVD:

$$A^T A = (U\sigma V^T)^T (U\sigma V^T) = V\sigma^2 V^T$$

$$AA^T = (U\sigma V^T)(U\sigma V^T)^T = U\sigma^2 U^T$$

If A is a complex matrix, the decomposition becomes:

$$A = U\sigma V^H, \quad A^H A = V\sigma^2 V^H, \quad AA^H = U\sigma^2 U^H$$

For a real symmetric matrix such as:

$$A = \begin{bmatrix} x & a & b \\ a & y & c \\ b & c & x \end{bmatrix}$$

we have $A = A^T$, implying $AA^T = A^TA$, and the SVD reduces to eigendecomposition:

$$A = P\sigma P^H \quad \text{or} \quad A = \sigma P^{-1}$$

where $PP^T = I$, and the columns of P are the eigenvectors $(v_1, v_2, ..., v_n)$ of A, with corresponding eigenvalues $(\lambda_1, \lambda_2, ..., \lambda_n)$.

Higher powers of A can be derived as:

$$A^2 = P\sigma^2P^{-1}, \quad A^3 = P\sigma^3P^{-1}, \quad A^4 = P\sigma^4P^{-1}$$

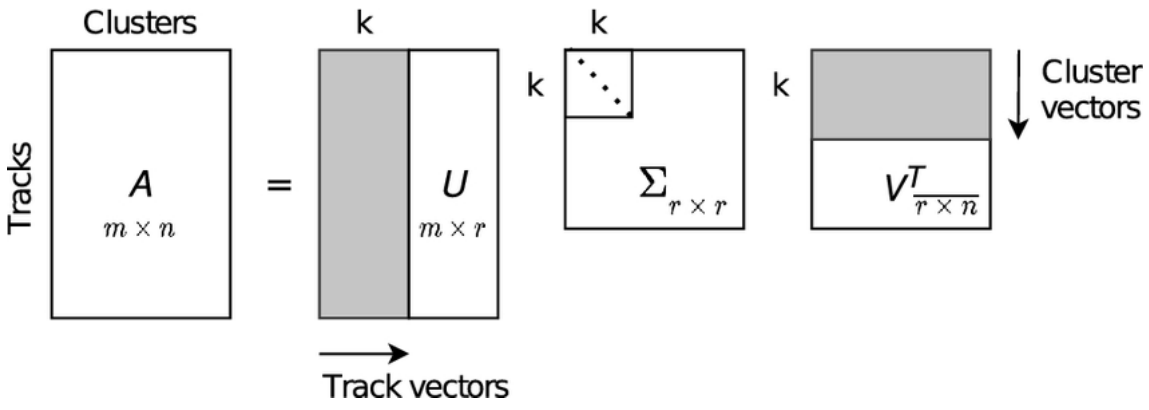


FIGURE 1.2: Approximated SVD [2]

1.2 SVD in Image Compression

The SVD representation of a matrix A as $A = U\sigma V^T$ enables efficient storage and reconstruction of image data. The singular values in σ are typically sorted in descending order:

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$$

where $r = \text{rank}(\sigma) = \text{rank}(A)$, and r is the number of non-zero singular values.

The total data needed to store the full SVD of an $m \times n$ matrix is:

$$m^2 + m + n^2$$

To reduce storage, a truncated SVD approximation retains only the top k singular values, resulting in:

- The first k columns of U and V
- The upper-left $k \times k$ block of σ

The new storage requirement becomes:

$$mk + k + kn$$

This compresses the data size to minimal levels while keeping the essential geometrical elements of the picture intact. The quality of the constructed image increases with greater k, but this requires additional space.

The referred method achieves such results and those results depend on the amount of singular values kept for the representation. An instance of the SVD approximation used in compression is depicted in Figure 1.2.

Chapter 2

Hardware Blocks in SVD

2.1 Floating Point Arithmetic: Addition, Subtraction, and Multiplication

IEEE single precision corresponds to the a 32-bit floating point number, which is offered along with a moderate compromise on the accuracy slant of the representation vs the amount of memory it occupies. These formats are extensively used for scientific calculations, computer graphics, and machine learning tasks.

A single-precision floating-point number consists:

- **Sign Bit (1 bit):** Shows the sign of the numerical value (0 if numerically positive, and 1 if numerically negative).
- **Exponent (8 bits):** Denotes the range for magnitude of the value as biased.
- **Significand (23 bits):** Holds the fractional portion of the number's representation excluding the leading one

Table 2.1: Single-Precision Floating-Point Representation

Sign (S)	**Exponent (E)**	**Significand (f)**
1 bit	8 bits	23 bits

The value of a single-precision floating-point number is calculated as:

$$\text{Value} = (-1)^S \cdot 2^{(E-127)} \cdot (1.f)$$

2.1.1 Single-Precision Addition and Subtraction

In floating-point addition or subtraction, the sign, exponent, and significand of each operand must be extracted. The smaller exponent is aligned to the larger by right-shifting the corresponding significand. Based on the signs of the numbers, the significands are then either added or subtracted.

After computing the result, it is normalized to fit the IEEE format, rounded, and packed back into a 32-bit representation. Special cases such as zero, infinity, and NaN (Not a Number) are also handled during this process.

2.1.2 Single-Precision Multiplication

The steps provided below detail the process of multiplying two floating point numbers:

- Retrieve the sign (S_x, S_y), exponent (E_x, E_y), and significand (f_x, f_y).
- $S_r = S_x \oplus S_y$ to calculate the sign.
- $E_r = (E_x + E_y) - 127$ to calculate the exponent.
- $M_r = (1.f_x) \cdot (1.f_y)$ to multiply the significands (along with the implicit leading one).

The product is shifted right and E_r is incremented if $M_r \geq 2$, otherwise, if $M_r < 1$, , then the product is left-shifted and E_r is decremented.After, modify the resulting significand by rounding it, then encode back into IEEE 754.

2.2 Rotation of a Point

Figure 2.1 illustrates the transformation of a point $V(x, y)$ rotated into a new position $V'(x', y')$.

$$x = r\cos(\phi), \quad y = r\sin(\phi)$$

$$x' = x\cos(\theta) - y\sin(\theta), \quad y' = x\sin(\theta) + y\cos(\theta)$$

This transformation can be expressed in matrix form as:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

The SVD decomposition incorporates these rotational matrices as follows:

$$\begin{bmatrix} w_1 & 0 \\ 0 & w_2 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} \cos(\phi) & \sin(\phi) \\ -\sin(\phi) & \cos(\phi) \end{bmatrix}$$

Here, the left and right rotation matrices are:

$$\text{Left: } \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}, \quad \text{Right: } \begin{bmatrix} \cos(\phi) & \sin(\phi) \\ -\sin(\phi) & \cos(\phi) \end{bmatrix}$$

Expanding the product yields:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} w_1\cos(\theta)\cos(\phi) + w_2\sin(\theta)\sin(\phi) & -w_1\cos(\theta)\sin(\phi) + w_2\sin(\theta)\cos(\phi) \\ -w_1\sin(\theta)\cos(\phi) + w_2\cos(\theta)\sin(\phi) & w_2\sin(\theta)\sin(\phi) + w_2\cos(\theta)\cos(\phi) \end{bmatrix}$$

From this, we define:

$$2e = a + d = (w_1 + w_2)\cos(\theta - \phi)$$
$$2f = a - d = (w_1 - w_2)\cos(\theta + \phi)$$
$$2g = b + c = (w_2 - w_1)\sin(\theta + \phi)$$
$$2h = b - c = (w_1 + w_2)\sin(\theta - \phi)$$

$$\tan(\theta - \phi) = \frac{h}{e}, \quad \tan(\theta + \phi) = \frac{-g}{f}$$

Therefore:

$$\theta - \phi = \tan^{-1}\left(\frac{h}{e}\right), \quad \theta + \phi = \tan^{-1}\left(\frac{-g}{f}\right)$$

These values can be determined in hardware using a CORDIC (Coordinate Rotation Digital Computer) unit [2], which computes trigonometric functions efficiently without requiring multiplication.

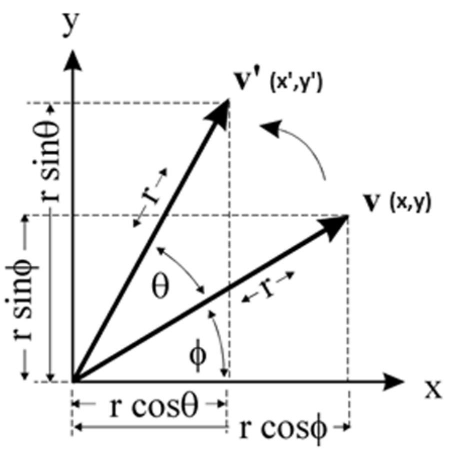


FIGURE 2.1: Rotation of a point V to V' [3]

Special Case: Symmetric Matrix

If the given matrix A is symmetric:

$$A = \begin{bmatrix} a_{pp} & a_{pq} \\ a_{pq} & a_{qq} \end{bmatrix}, \quad \text{where } a_{pq} = a_{qp}$$

Then the rotation angles simplify to:

$$\theta - \phi = \tan^{-1}(0) = 0, \quad \theta + \phi = \tan^{-1}\left(\frac{-2a_{pq}}{a_{pp} - a_{qq}}\right)$$

Chapter 3

Implementations of SVD

Our Proposed Design framework embeds two key implementations namely Jacobi SVD (or Two Sided SVD) and Hestenes SVD (or One Sided SVD)

3.1 Jacobi SVD (or Two Sided SVD)

The main principle of the Jacobi method [5] is that matrix A is multiplied by Jacobi Rotation matrices successively to nullify the off-diagonal elements (or non-diagonal elements). Multiplication is done from the right and left sides of matrix A. Left side Jacobi matrices are denoted as J_l and right-side Jacobi matrices are denoted as J_r. Multiplication by the Jacobi matrices is shown below.
$S = J_{l_k} J_{l(k-1)} \dots J_{l_1} J_{l_0} A J_{r_0} J_{r_1} \dots J_{r(k-1)} J_{r_k}$. Here, k represents the number of iterations after which we get the singular values (or the diagonal matrix). The orthogonal matrices can be computed as follows. $U = J_{l_k} J_{l(k-1)} \dots J_{l_1} J_{l_0}$ and $V = J_{r_0} J_{r_1} \dots J_{r(k-1)} J_{r_k}$

Here, all the rotation matrices such as $J_{lk}, J_{l(k-1)}, \dots, J_{l1}, J_{l0}, J_{lk}, J_{l(k-1)}, \dots, J_{l1}$ and J_{l0} are orthogonal. This method is called double sided as matrix A is multiplied by Jacobi rotation matrices from both sides. Multiplication by left Jacobi matrix means rotation of rows of A and multiplication by right Jacobi matrix means rotation of columns of matrix A.

In one iteration of a sweep two elements are nullified and there are 6 iterations in a sweep. The Jacobi left rotation matrix [6] [7] is shown in the below Figure 3.1.

$$J_{ij} = \begin{cases} J_{pp} = \cos(\theta) \\ J_{qp} = -\sin(\theta) \\ J_{pq} = \sin(\theta) \\ J_{qq} = \cos(\theta) \\ J_{ii} = 1 \,; i \neq p,q; \\ J_{ij} = 0 \,; otherwise \end{cases} \rightarrow \begin{bmatrix} 1 & & & & 0 \\ \ddots & & \vdots & & \iddots \\ & \cos(\theta) \cdots & 0 & \cdots -\sin(\theta) & \\ \cdots & 0 & \cdots 1 \cdots & 0 & \cdots \\ & \sin(\theta) \cdots & 0 & \cdots \cos(\theta) & \\ \iddots & & \vdots & & \ddots \\ 0 & & & & 1 \end{bmatrix}$$

FIGURE 3.1: Jacobi left rotation matrix used in two sided SVD

Let an matrix A = $\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix}$. Now the objective is to find the diagonal matrix of A using two-sided Jacobi rotation.

Step-1: Let us assume that a_{12} is the largest off-diagonal element of A, then

$$A^0 = J_{l0} \cdot A \cdot J_{r0} = \begin{bmatrix} c_l & -s_l & 0 & 0 & 0 \\ s_l & c_l & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot A \cdot \begin{bmatrix} c_r & -s_r & 0 & 0 & 0 \\ s_r & c_r & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

where $s_l = \sin(\theta), c_l = \cos(\theta), s_r = \sin(\phi), c_r = \cos(\phi)$

$\theta - \phi = \tan^{-1}\left(\frac{a_{12}-a_{21}}{a_{11}+a_{22}}\right)$ and $\theta + \phi = -\tan^{-1}\left(\frac{a_{12}+a_{21}}{a_{11}-a_{22}}\right)$

Step-2: Let us assume that a_{23}^0 is the largest off-diagonal element of A^0, then

$$A^1 = J_{l1} \cdot A^0 \cdot J_{r1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & c_l & -s_l & 0 & 0 \\ 0 & s_l & c_l & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot A^0 \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & c_r & -s_r & 0 & 0 \\ 0 & s_r & c_r & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$\theta - \phi = \tan^{-1}\left(\frac{a_{23}-a_{32}}{a_{22}+a_{33}}\right)$ and $\theta + \phi = -\tan^{-1}\left(\frac{a_{23}+a_{32}}{a_{22}-a_{33}}\right)$

Step-3: Let us assume that a^1_{13} is the largest off-diagonal element of A^1, then

$$A^2 = J_{l2} \cdot A^1 \cdot J_{r2} = \begin{bmatrix} c_l & 0 & -s_l & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ s_l & 0 & c_l & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot A^1 \cdot \begin{bmatrix} c_r & 0 & -s_r & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ s_r & 0 & c_r & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$\theta - \phi = \tan^{-1}\left(\frac{a_{13}-a_{31}}{a_{11}+a_{33}}\right)$ and $\theta + \phi = -\tan^{-1}\left(\frac{a_{13}+a_{31}}{a_{11}-a_{33}}\right)$

Step-4: Let us assume that a^2_{14} is the largest off-diagonal element of A^2, then

$$A^3 = J_{l3} \cdot A^2 \cdot J_{r3} = \begin{bmatrix} c_l & 0 & 0 & -s_l & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ s_l & 0 & 0 & c_l & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot A^2 \cdot \begin{bmatrix} c_r & 0 & 0 & -s_r & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ s_r & 0 & 0 & c_r & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$\theta - \phi = \tan^{-1}\left(\frac{a_{14}-a_{41}}{a_{11}+a_{44}}\right)$ and $\theta + \phi = -\tan^{-1}\left(\frac{a_{14}+a_{41}}{a_{11}-a_{44}}\right)$

Step-5: Let us assume that a^3_{15} is the largest off-diagonal element of A^3, then

$$A^4 = J_{l4} \cdot A^3 \cdot J_{r4} = \begin{bmatrix} c_l & 0 & 0 & 0 & -s_l \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ s_l & 0 & 0 & 0 & c_l \end{bmatrix} \cdot A^3 \cdot \begin{bmatrix} c_r & 0 & 0 & 0 & -s_r \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ s_r & 0 & 0 & 0 & c_r \end{bmatrix}$$

$\theta - \phi = \tan^{-1}\left(\frac{a_{15}-a_{51}}{a_{11}+a_{55}}\right)$ and $\theta + \phi = -\tan^{-1}\left(\frac{a_{15}+a_{51}}{a_{11}-a_{55}}\right)$

Step-6: Let us assume that a^4_{24} is the largest off-diagonal element of A^4, then

$$A^5 = J_{l5} \cdot A^4 \cdot J_{r5} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & c_l & 0 & -s_l & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & s_l & 0 & c_l & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot A^3 \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & c_r & 0 & -s_r & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & s_l & 0 & c_r & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$\theta - \phi = \tan^{-1}\left(\frac{a_{24}-a_{42}}{a_{22}+a_{44}}\right)$ and $\theta + \phi = -\tan^{-1}\left(\frac{a_{24}+a_{42}}{a_{22}-a_{44}}\right)$

Step-7: Let us assume that a_{25}^5 is the largest off-diagonal element of A^5, then

$$A^6 = J_{l6} \cdot A^5 \cdot J_{r6} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & c_l & 0 & 0 & -s_l \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & s_l & 0 & 0 & c_l \end{bmatrix} \cdot A^5 \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & c_r & 0 & 0 & -s_r \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & s_r & 0 & 0 & c_r \end{bmatrix}$$

$\theta - \phi = \tan^{-1}\left(\frac{a_{25}-a_{52}}{a_{22}+a_{55}}\right)$ and $\theta + \phi = -\tan^{-1}\left(\frac{a_{25}+a_{52}}{a_{22}-a_{55}}\right)$

Step-8: Let us assume that a_{35}^6 is the largest off-diagonal element of A^6, then

$$A^7 = J_{l7} \cdot A^6 \cdot J_{r7} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & c_l & 0 & -s_l \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & s_l & 0 & c_l \end{bmatrix} \cdot A^6 \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & c_r & 0 & -s_r \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & s_r & 0 & c_r \end{bmatrix}$$

$\theta - \phi = \tan^{-1}\left(\frac{a_{35}-a_{53}}{a_{33}+a_{55}}\right)$ and $\theta + \phi = -\tan^{-1}\left(\frac{a_{35}+a_{53}}{a_{33}-a_{55}}\right)$

Step-9: Let us assume that a_{34}^7 is the largest off-diagonal element of A^7, then

$$A^8 = J_{l8} \cdot A^7 \cdot J_{r8} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & c_l & -s_l & 0 \\ 0 & 0 & s_l & c_l & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot A^7 \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & c_r & -s_r & 0 \\ 0 & 0 & s_r & c_r & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$\theta - \phi = \tan^{-1}\left(\frac{a_{34}-a_{43}}{a_{33}+a_{44}}\right)$ and $\theta + \phi = -\tan^{-1}\left(\frac{a_{34}+a_{43}}{a_{33}-a_{44}}\right)$

Step-10: Let us assume that a_{45}^8 is the largest off-diagonal element of A^8, then

$$A^9 = J_{l9} \cdot A^8 \cdot J_{r9} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & c_l & -s_l \\ 0 & 0 & 0 & s_l & c_l \end{bmatrix} \cdot A^8 \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & c_r & -s_r \\ 0 & 0 & 0 & s_r & c_r \end{bmatrix}$$

$\theta - \phi = \tan^{-1}\left(\frac{a_{45}-a_{54}}{a_{44}+a_{55}}\right)$ and $\theta + \phi = -\tan^{-1}\left(\frac{a_{45}+a_{54}}{a_{44}-a_{55}}\right)$

If all the off-diagonal elements are zero in any one of the steps 1 to 10, then this recursive procedure will be stopped.

3.2 Hestenes SVD (or One Sided SVD):

The main principle Hestenes method or one sided SVD [3] is to find only only otation matrix for the singular value decomposition. Let A = U.Σ.V^T, where A is the given matrix which is to be decomposed in to U, Σ, and V^T using SVD. Here, U and V are orthogonal matrices. The diagonal matrix is Σ. Let A.V = U.Σ.V^T.V = U.Σ.I = U.Σ. Now the objective is to find V. Let W = A.V(=U.Σ).

We should find W such that all the columns of W should be orthogonal to each other. It can be done by finding the value of $A^{(k+1)} = A^{(k)}.Q^{(k)}$, where $A^{(0)}$ = A (the given matrix which is to be decomposed). Finding W includes several sweeps. Each sweep includes a several iterations. The rotation matrix is Q^k, where $0 \leq$ k $\leq$ M. Here, M is the number of iterations required to find W. Now, the objective is to find the rotation matrix $Q^{(k)}$ such that the corresponding two columns of $A^{(k+1)}$ should be orthogonal.

Let an matrix $A(0) = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix}$

Sweep 1: Our interest is to make the columns i = 1 and 2 of $A^{(0)}$ to be orthogonal.

$$A^{(1)} = A^{(0)}.Q^{(0)} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix} \cdot \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 & 0 & 0 \\ -\sin(\theta) & \cos(\theta) & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Here, the columns i = 1 and 2 of $A^{(1)}$ are orthogonal.

$$A^{(2)} = A^{(1)}.Q^{(1)} = A^{(1)}. \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Here, the columns i = 1 and 3 of $A^{(2)}$ are orthogonal.

$$A^{(3)} = A^{(2)}.Q^{(2)} = A^{(2)}. \begin{bmatrix} \cos(\theta) & 0 & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Here, the columns i = 1 and 4 of $A^{(3)}$ are orthogonal.

$$A^{(4)} = A^{(3)}.Q^{(3)} = A^{(3)}. \begin{bmatrix} \cos(\theta) & 0 & 0 & 0 & \sin(\theta) \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -\sin(\theta) & 0 & 0 & 0 & \cos(\theta) \end{bmatrix}$$

Here, the columns i = 1 and 5 of $A^{(4)}$ are orthogonal. Here, the columns $i = 1$ and 5 of $A^{(4)}$ are orthogonal.

The columns $i = 2$ and 3 of $A^{(5)} = A^{(4)}Q^{(4)}$ are orthogonal.

The columns $i = 2$ and 4 of $A^{(6)} = A^{(5)}Q^{(5)}$ are orthogonal.

The columns $i = 2$ and 5 of $A^{(7)} = A^{(6)}Q^{(6)}$ are orthogonal.

The columns $i = 3$ and 4 of $A^{(8)} = A^{(7)}Q^{(7)}$ are orthogonal.

The columns $i = 3$ and 5 of $A^{(9)} = A^{(8)}Q^{(8)}$ are orthogonal.

The columns $i = 4$ and 5 of $A^{(10)} = A^{(9)}Q^{(9)}$ are orthogonal.

Sweep-2:

The columns $i = 1$ and 2 of $A^{(1)} = A^{(0)}Q^{(0)}$ are orthogonal, where $A^{(0)} = A^{(10)}$ of Sweep-1.

The columns $i = 1$ and 3 of $A^{(2)} = A^{(1)}Q^{(1)}$ are orthogonal.

The columns $i = 1$ and 4 of $A^{(3)} = A^{(2)}Q^{(2)}$ are orthogonal.

The columns $i = 1$ and 5 of $A^{(4)} = A^{(3)}Q^{(3)}$ are orthogonal.

The columns $i = 2$ and 3 of $A^{(5)} = A^{(4)}Q^{(4)}$ are orthogonal.

The columns $i = 2$ and 4 of $A^{(6)} = A^{(5)}Q^{(5)}$ are orthogonal.

The columns $i = 2$ and 5 of $A^{(7)} = A^{(6)}Q^{(6)}$ are orthogonal.

The columns $i = 3$ and 4 of $A^{(8)} = A^{(7)}Q^{(7)}$ are orthogonal.

The columns $i = 3$ and 5 of $A^{(9)} = A^{(8)}Q^{(8)}$ are orthogonal.

The columns $i = 4$ and 5 of $A^{(10)} = A^{(9)}Q^{(9)}$ are orthogonal.

Value of θ:

Let $A^{(0)} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix}$. For the columns i = 1 and 2,

$\alpha_{ii} = a_{11}.a_{11} + a_{21}.a_{21} + a_{31}.a_{31} + a_{41}.a_{41} + a_{51}.a_{51}$ (dot product)

$\alpha_{jj} = a_{12}.a_{12} + a_{22}.a_{22} + a_{32}.a_{32} + a_{42}.a_{42} + a_{52}.a_{52}$ (dot product)

$\gamma_{ij} = a_{11}.a_{12} + a_{21}.a_{22} + a_{31}.a_{32} + a_{41}.a_{42} + a_{51}.a_{52}$ (dot product)

$\xi = \frac{\alpha_{jj}-\alpha_{ii}}{2\gamma_{ij}}$

$\tan(\theta) = \frac{signof(\xi)}{|\xi|+\sqrt{1+\xi^2}}$

$\cos(\theta) = \frac{1}{\sqrt{(1+\tan(\theta)^2)}}$

$\sin(\theta) = \tan(\theta).\cos(\theta)$

In each iteration of each sweep, the values of $\cos(\theta)$ and $\sin(\theta)$ should be known.

In an n×n matrix, a sweep includes $\frac{n(n-1)}{2}$ interations. At the end of each sweep, we need to find the value of V = $Q^0.Q^1.\dots.Q^{\frac{n(n-1)}{2}}$. In other words, V is the multiplication of all the rotation matrices. Then we need to find whether all the columns of W = A.V are orthogonal to each other or not. If not, then the next sweep with the input from the output of previous sweep has to be started. So, the total number of iterations to find W is M = S.$\frac{n(n-1)}{2}$, where S is the total number of sweeps required. If W = A.V has only the orthogonal columns, then the sweep has to be stopped.

Norm of the matrix:

Let W = $\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$, where all the columns are orthogonal to each other.

Norm of W at column i to be 1 = $\sigma_1 = \sqrt{(a11)^2 + (a_{21})^2 + (a_{31})^2}$

Norm of W at column i to be 2 =$\sigma_2 = \sqrt{(a_{12})^2 + (a_{22})^2 + (a_{32})^2}$

Norm of W at column i to be 3 =$\sigma_3 = \sqrt{(a_{13})^2 + (a_{23})^2 + (a_{33})^2}$

The diagonal matrix $\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix}$

The orthogonal matrix U = $\frac{W}{\Sigma}$ (point-wise division)

Here, U = $\begin{bmatrix} \frac{a_{11}}{\sigma_1} & \frac{a_{12}}{\sigma_2} & \frac{a_{31}}{\sigma_3} \\ \frac{a_{21}}{\sigma_1} & \frac{a_{22}}{\sigma_2} & \frac{a_{32}}{\sigma_3} \\ \frac{a_{13}}{\sigma_1} & \frac{a_{23}}{\sigma_2} & \frac{a_{33}}{\sigma_3} \end{bmatrix}$

Finally, A = U.Σ.V^T

3.3 Architecture (Module1):

$\alpha_{ii} = a_{11}.a_{11} + a_{21}.a_{21}$ (dot product)

$\alpha_{jj} = a_{12}.a_{12} + a_{22}.a_{22}$ (dot product)

$\gamma_{ij} = a_{11}.a_{21} + a_{12}.a_{22}$ (dot product)

Left shifting for getting 2*γ_{ij}

$\xi = \frac{\alpha_{jj} - \alpha_{ii}}{2\gamma_{ij}}$

For calculating ξ, difference of alpha22 and alpha11 and dividing it by 2*γ_{ij}

Next, finding $\xi^2, 1 + \xi^2, signofeta, absoluteeta$

$Fromthe\xi$ value we have

$\tan(\theta) = \frac{signof(\xi)}{|\xi| + \sqrt{1+\xi^2}}$

$\cos(\theta) = \frac{1}{\sqrt{(1+\tan(\theta)^2)}}$

$\sin(\theta) = \tan(\theta).\cos(\theta)$

The Module is as shown in the below Figure 3.2

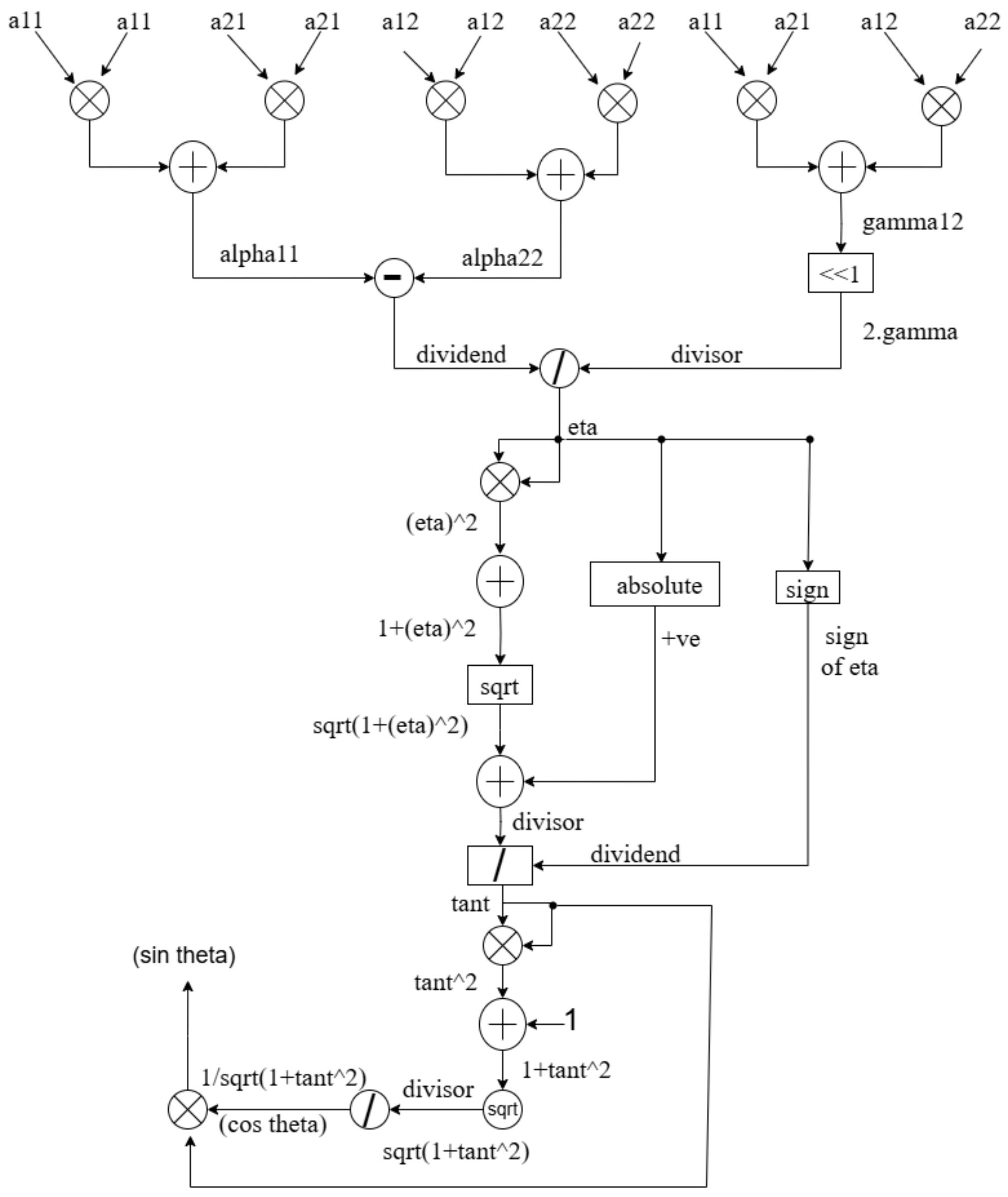


FIGURE 3.2: Architecture of module-1 used in 2x2 for One-sided Jacobi SVD

3.4 Architecture (Module2:)

Getting the outputs of the module 1: We find the 1st rotation matrix:

b11 = a11*cos(θ) + a12*(-sin(θ))

b21 = a21*cos(θ) + a22*(-sin(θ))

b12 = a21*cos(θ) + a12*(sin(θ))

b22 = a21*sin(θ) + a22*(cos(θ))

B = A*V that has orthogonal columns, where the sweep stops.

Finding the Norm of the B matrix gives the (Σ matrix.

Norm of W at column 1 = $\sigma_1 = \sqrt{(a11)^2 + (a_{21})^2}$

Norm of W at column 2 =$\sigma_2 = \sqrt{(a_{12})^2 + (a_{22})^2}$

The orthogonal matrix U = $\frac{W}{\Sigma}$

The Module is as shown in the below Figure 3.3

The Overall Module is shown in the Figure 3.4

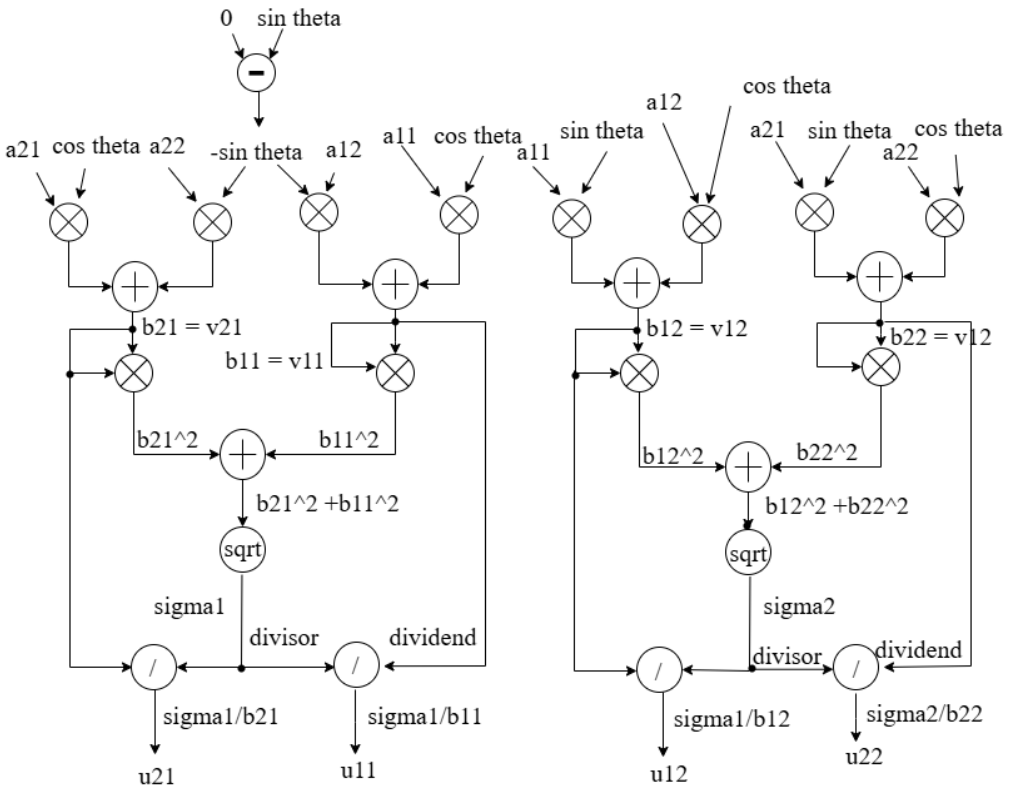


Figure 3.3: Architecture of module-2 used in 2x2 for One-sided Jacobi SVD

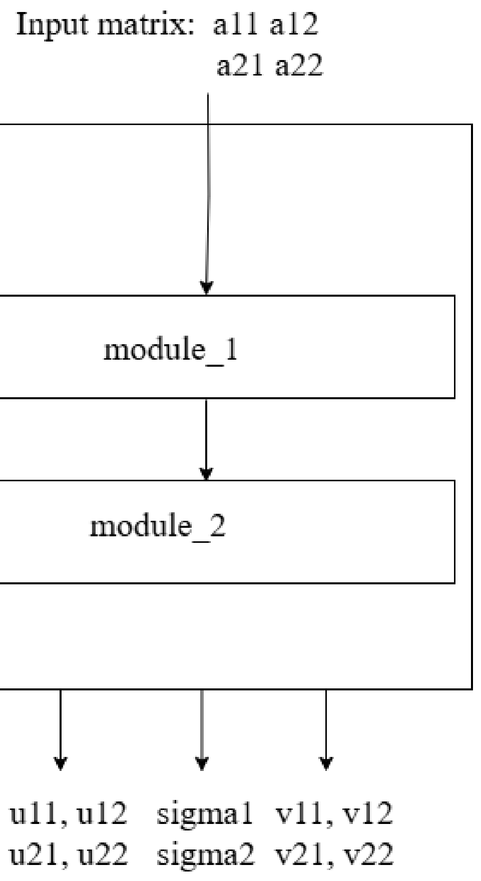
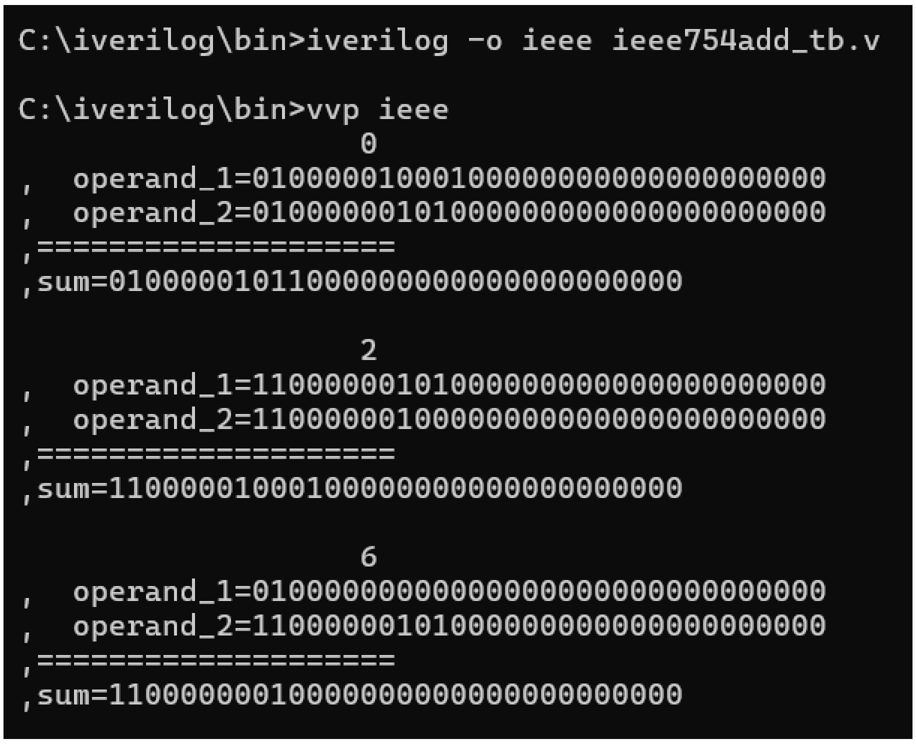


FIGURE 3.4: The Co-processor used in 2x2 One-Sided SVD

Chapter 4

Results and Discussion

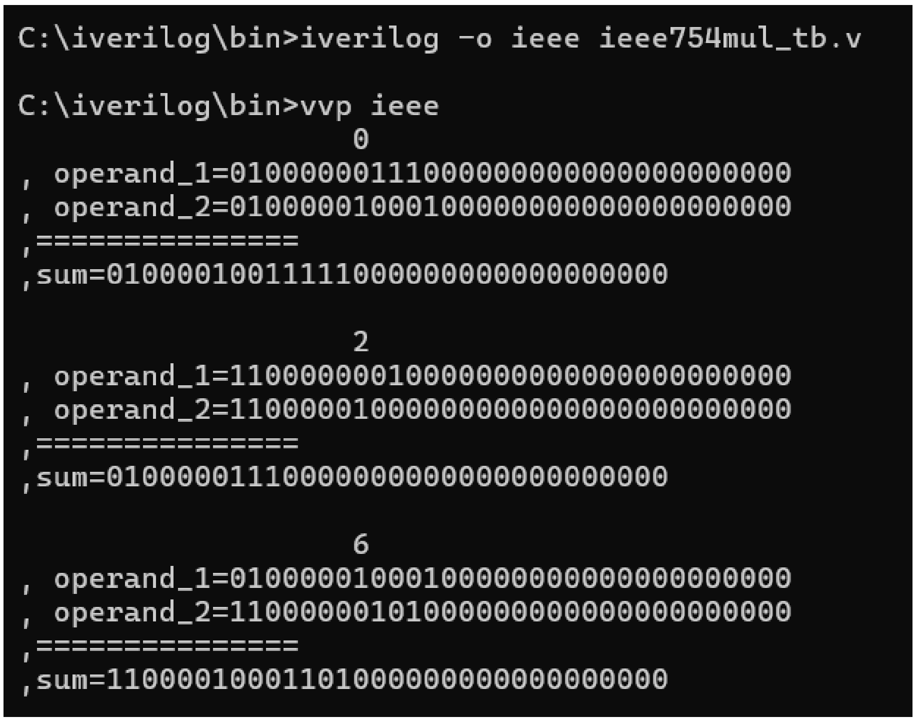
4.1 Floating Point Addition



```
C:\iverilog\bin>iverilog -o ieee ieee754add_tb.v

C:\iverilog\bin>vvp ieee
                   0
,  operand_1=01000001000100000000000000000000
,  operand_2=01000001010000000000000000000000
,====================
,sum=01000001011000000000000000000000

                   2
,  operand_1=11000001010000000000000000000000
,  operand_2=11000001000000000000000000000000
,====================
,sum=11000001000100000000000000000000

                   6
,  operand_1=01000000000000000000000000000000
,  operand_2=11000001010000000000000000000000
,====================
,sum=11000000010000000000000000000000
```

Figure 4.1: Output for IEEE 754 single precision Floating Point Addition

4.2 Floating Point Multiplication

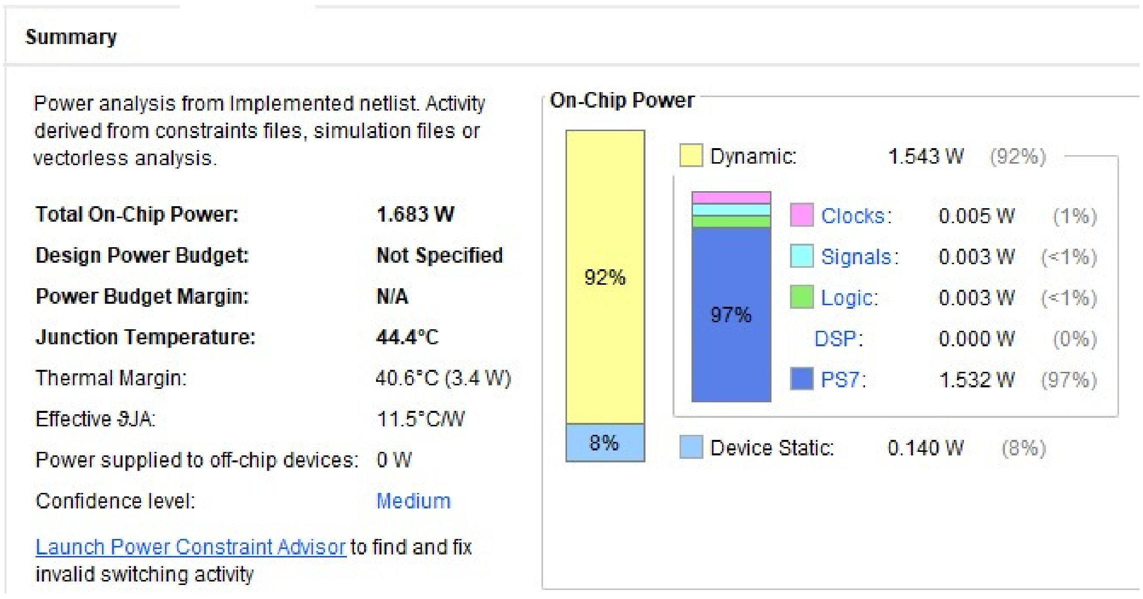


```
C:\iverilog\bin>iverilog -o ieee ieee754mul_tb.v

C:\iverilog\bin>vvp ieee
                   0
, operand_1=01000000111000000000000000000000
, operand_2=01000001000100000000000000000000
,===============
,sum=01000010011111000000000000000000

                   2
, operand_1=11000000010000000000000000000000
, operand_2=11000001000000000000000000000000
,===============
,sum=01000001110000000000000000000000

                   6
, operand_1=01000001000100000000000000000000
, operand_2=11000000101000000000000000000000
,===============
,sum=11000010001101000000000000000000
```

FIGURE 4.2: Output for IEEE 754 single precision Floating Point Multiplication

4.3 One-Sided Jacobi

For Singular Value Decomposition (SVD) the One-sided Jacobi method was used to calculate the singular values of a given 4x4 matrix A_1 , with iterations of rotations until off-diagonal elements have become zero. By computing the square roots of the eigenvalues of $A_1^T A_1$ the resulting singular values are [9.0687 4.6708 3.9697 0.4282]. These values give insight into the matrix's properties and provide a measure of the rank and the differences in various directions of data.



Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

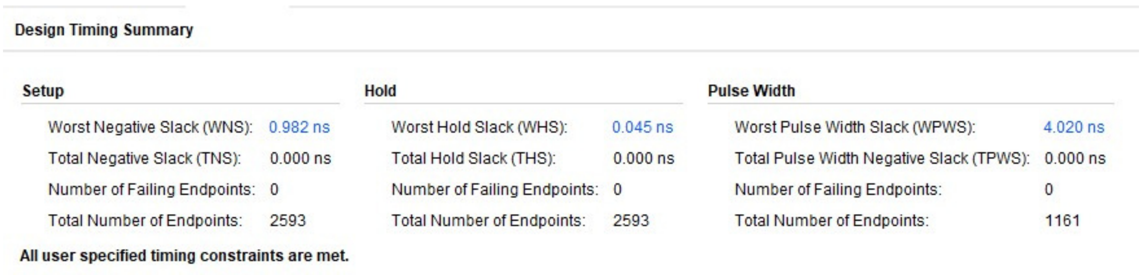
Total On-Chip Power:	**1.683 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**44.4°C**
Thermal Margin:	40.6°C (3.4 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

🟨	Dynamic:	1.543 W	(92%)
🟪	Clocks:	0.005 W	(1%)
🟦	Signals:	0.003 W	(<1%)
🟩	Logic:	0.003 W	(<1%)
	DSP:	0.000 W	(0%)
🟦	PS7:	1.532 W	(97%)
🟦	Device Static:	0.140 W	(8%)

FIGURE 4.3: Power Analysis of IEEE 754 Single Precision Floating Point Multiplication using Zynq 7000 FPGA



Design Timing Summary

Setup		**Hold**		**Pulse Width**	
Worst Negative Slack (WNS):	0.982 ns	Worst Hold Slack (WHS):	0.045 ns	Worst Pulse Width Slack (WPWS):	4.020 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	2593	Total Number of Endpoints:	2593	Total Number of Endpoints:	1161

All user specified timing constraints are met.

FIGURE 4.4: Timing Summary of IEEE 754 Single Precision Floating Point Multiplication using Zynq 7000 FPGA

Jacobi method: the method could convert the original matrix into a diagonal form, with the singular values appearing along the diagonal. However, Jacobi methods may not be the most efficient algorithm for larger matrices than those of QR decomposition methods. The obtained Matlab results are shown in Figure 4.5.

4.4 Two-Sided Jacobi

The Two-Sided Jacobi Method for SVD is an improvement from one-sided approaches which applies orthogonal rotations to the matrix A_1 and to its transpose A_1^T to try to diagonalize the matrix $A_1^T A_1$ to obtain the diagonal form with only the singular values on the diagonal of the matrix after making off-diagonal elements zero. It is more general since it also computes both the left and the right singular vectors. Though this two-sided Jacobi method provides high accuracy, for large matrices, it is computational expensive, which means that other algorithms like QR decomposition are more efficient. The Obtained Matlab results are shown in Figure 4.6.

Chapter 5

Implementation Methodology

5.1 Flow of Design in Xilinx Vivado EDA tool

This section covers the flow of design in Xilinx Vivado EDA tool. The design flow covers everything from the creation of custom IP to the bit stream generation. In this section, two design flows are discussed, one for the proposed design and the other for the proposed design including partial reconfiguration.

5.1.1 Design Flow

1. Verilog codes are written and are simulated in the Xilinx Vivado EDA tool.
2. Test and verify the simulation results and if they are incorrect, repeat Step 1.
3. Create a custom Interface Peripheral (IP)
4. Instantiate the created design in the created IP where the inputs and outputs of the design are stored in the AXI-Slave registers
5. Review and Package the IP
6. Create the block design for various implementation as shown in Figure 5.1. Validate the block designs and wrap the block designs with HDL wrapper.

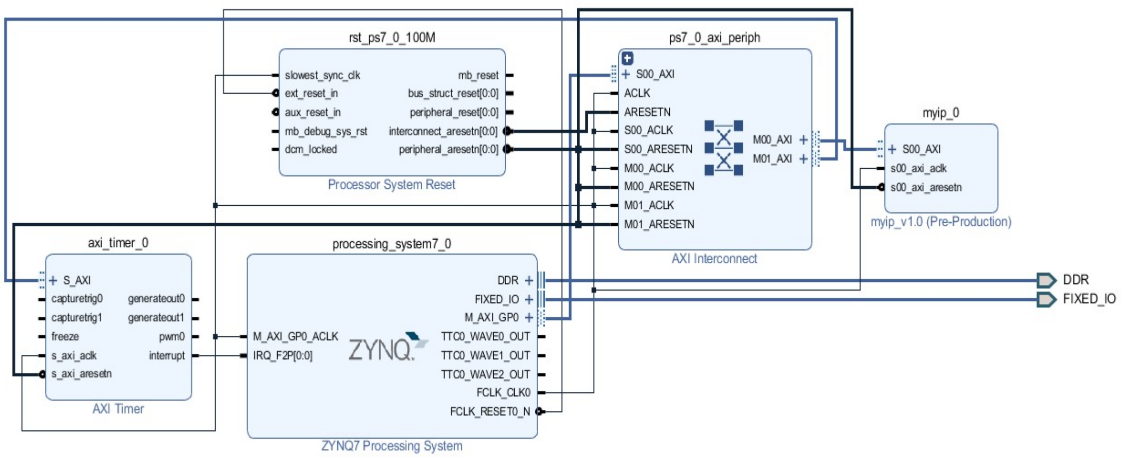
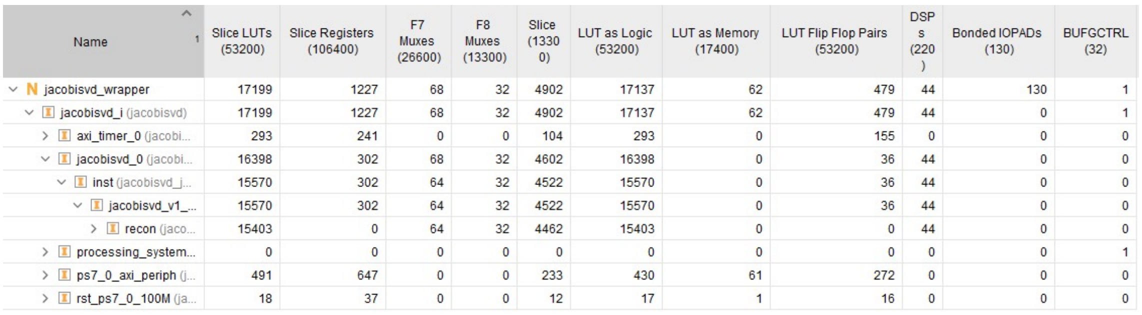


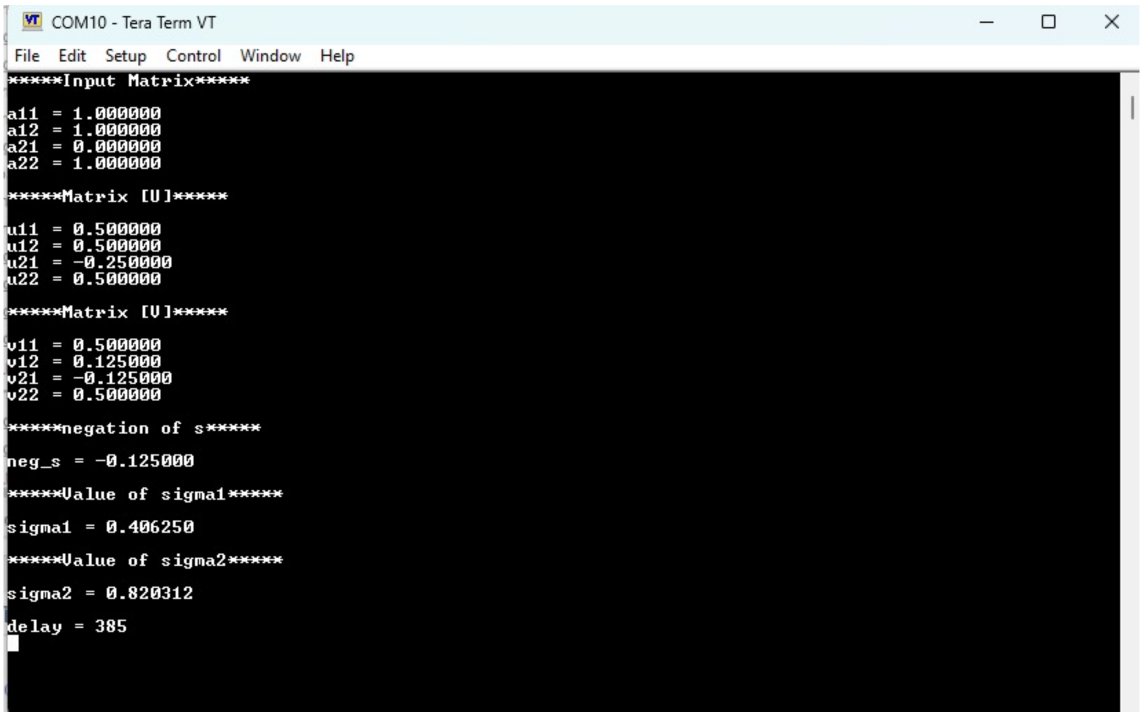
FIGURE 5.1: Custom IP integration diagram of 2x2 One-sided SVD of using Zynq 7000 FPGA with Xilinx Vivado

7. Generate Bitstream for the wrapped block design.
8. If the design does not meet timing requirements, use the concept of pipelining and repeat from Step-1.
9. Export the hardware by including the bitstream file
10. Program the target FPGA with the generated bitstream file. A blue light glows on the FPGA, indicating that the device is programmed.
11. Launch Xilinx SDK directory and create a new application project.
12. Start the Hardware Software Co-design in Xilinx SDK platform.



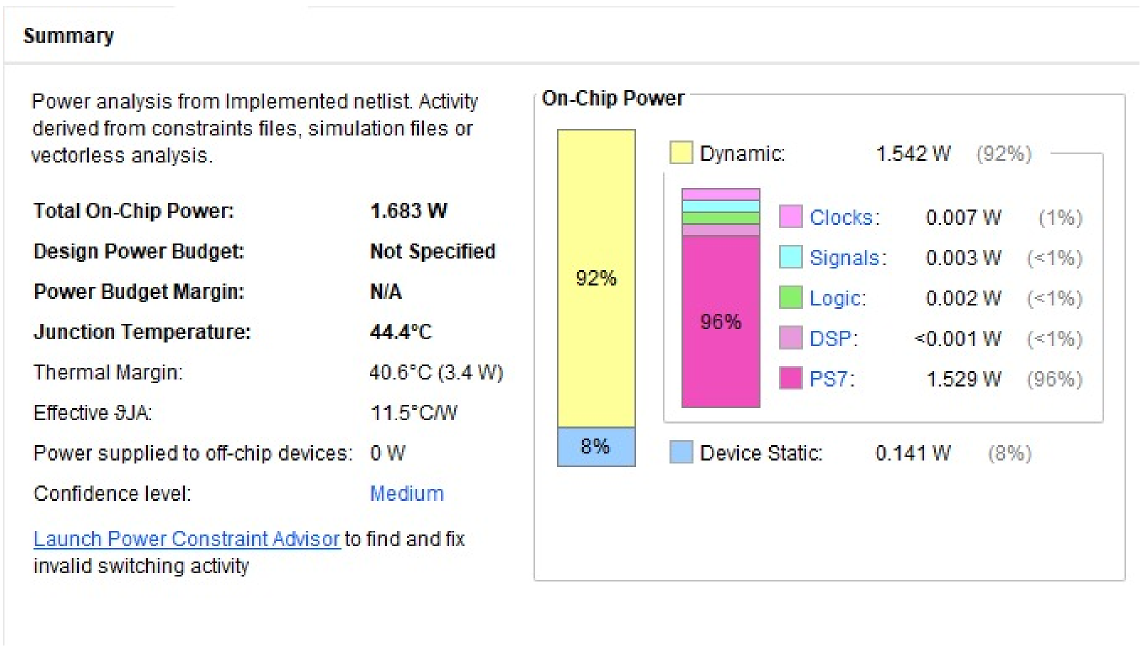
Name	Slice LUTs (53200)	Slice Registers (106400)	F7 Muxes (26600)	F8 Muxes (13300)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)	LUT Flip Flop Pairs (53200)	DSPs (220)	Bonded IOPADs (130)	BUFGCTRL (32)
jacobisvd_wrapper	17199	1227	68	32	4902	17137	62	479	44	130	1
jacobisvd_i (jacobisvd)	17199	1227	68	32	4902	17137	62	479	44	0	1
axi_timer_0 (jacobi...	293	241	0	0	104	293	0	155	0	0	0
jacobisvd_0 (jacobi...	16398	302	68	32	4602	16398	0	36	44	0	0
inst (jacobisvd_j...	15570	302	64	32	4522	15570	0	36	44	0	0
jacobisvd_v1_...	15570	302	64	32	4522	15570	0	36	44	0	0
recon (jaco...	15403	0	64	32	4462	15403	0	0	44	0	0
processing_system...	0	0	0	0	0	0	0	0	0	0	1
ps7_0_axi_periph (j...	491	647	0	0	233	430	61	272	0	0	0
rst_ps7_0_100M (ja...	18	37	0	0	12	17	1	16	0	0	0

Figure 5.2: Utilization Report of 2x2 One-sided SVD using Zynq 7000 FPGA



```
*****Input Matrix*****

a11 = 1.000000
a12 = 1.000000
a21 = 0.000000
a22 = 1.000000

*****Matrix [U]*****

u11 = 0.500000
u12 = 0.500000
u21 = -0.250000
u22 = 0.500000

*****Matrix [V]*****

v11 = 0.500000
v12 = 0.125000
v21 = -0.125000
v22 = 0.500000

*****negation of s*****

neg_s = -0.125000

*****Value of sigma1*****

sigma1 = 0.406250

*****Value of sigma2*****

sigma2 = 0.820312

delay = 385
```

Figure 5.3: Delay of 2x2 One-Sided Jacobi SVD using Zynq 7000 FPGA



Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.683 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**44.4°C**
Thermal Margin:	40.6°C (3.4 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:		1.542 W	(92%)
	Clocks:	0.007 W	(1%)
	Signals:	0.003 W	(<1%)
	Logic:	0.002 W	(<1%)
	DSP:	<0.001 W	(<1%)
	PS7:	1.529 W	(96%)
Device Static:		0.141 W	(8%)

FIGURE 5.4: Power Summary of 2x2 One-Sided Jacobi SVD using Zynq 7000 FPGA

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The goal of this project was to design and implement efficient VLSI architectures for Singular Value Decomposition SVD, concentrating on the Jacobi (two-sided) and Hestenes (one-sided) methods. The Jacobi method is typically more accurate than the Hestenes, but also more resource demanding. In particular, the project aimed at SVD realization analysis on FPGA platforms using Xilinx Vivado tools. This work improves the state of the art in Xilinx FPGAs in comparison to other competitors. The project examined both methods concerning their computational performance, accuracy, and resource utilization. Both approaches achieved accurate singular values but the one-sided Jacobi method was weaker in terms of hardware while the two-sided method was stronger for full decomposition including singular vectors.

The method combines performing different data representations through IP cores as building blocks with overwhelming results. The modular AXI interfaces allow for easy configuration and thorough testing without having to change underlying designs.

6.2 Future Work

There are multiple avenues of further improvement which can be pursued:

Parallelization and Pipelining: Adding more complete processing elements in addition to pipelined data paths greatly enhances throughput, especially with higher dimension matrices.

Partial Reconfiguration: Using FPGA partial reconfiguration can optimally switch between SVD variants or precision levels based on runtime requirements.

Support for Larger Matrices: Current implementations can be efficiently scaled to larger matrices through hierarchical decomposition techniques, as well as applied distributed processing.

Bibliography

[1] Cmu school of computer science. Accessed: 2024-11-23. [Online]. Available: https://www.cs.cmu.edu/~venkatg/teaching/CStheory-infoage/book-chapter-4.pdf

[2] J. Cavallaro, M. Keleher, R. Price, and G. Thomas, “Vlsi implementation of a cordic svd processor,” in *Proceedings., Eighth University/Government/Industry Microelectronics Symposium*, 1989, pp. 256–260.

[3] B. Zhou and R. Brent, “On parallel implementation of the one-sided jacobi algorithm for singular value decompositions,” in *Proceedings Euromicro Workshop on Parallel and Distributed Processing*, 1995, pp. 401–408.

[4] (2021) Zerobone.net. Accessed: 14.01.2021. [Online]. Available: https://zerobone.net/blog/cs/svd-image-compression/

[5] Digital system design. Accessed: 2023-03-08. [Online]. Available: https://digitalsystemdesign.in/svd-implementation-strategies/

[6] W. Ma, M. E. Kaye, D. M. Luke, and R. Doraiswami, “An fpga-based singular value decomposition processor,” in *2006 Canadian Conference on Electrical and Computer Engineering*, 2006, pp. 1047–1050.

[7] A. Ahmedsaid, A. Amira, and A. Bouridane, “Improved svd systolic array and implementation on fpga,” in *Proceedings. 2003 IEEE International Conference on Field-Programmable Technology (FPT) (IEEE Cat. No.03EX798)*, 2003, pp. 35–42.

[8] Y. Zhang, “An introduction to matrix factorization and factorization machines in recommendation system, and beyond,” pp. 2–4, 2022.

[9] A. Yürekli, C. Kaleli, and A. Bilge, "Alleviating the cold-start playlist continuation in music recommendation using latent semantic indexing," *International Journal of Multimedia Information Retrieval*, vol. 10, pp. 1–14, 2021.

[10] J. Blinn, "Consider the lowly 2 x 2 matrix," *IEEE Computer Graphics and Applications*, vol. 16, no. 2, pp. 82–88, 1996.

[11] M. Rahmati, M. S. Sadri, and M. A. Naeini, "Fpga based singular value decomposition for image processing applications," in *2008 International Conference on Application-Specific Systems, Architectures and Processors*, 2008, pp. 185–190.

[12] H. Snopce, A. Aliu, and A. Luma, "The block jacobi method for the svd-a parallel approach," in *2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, 2020, pp. 1–5.

[13] A. Shiri and G. Khosroshahi, "An fpga implementation of singular value decomposition," 2019, pp. 416–422.

[14] Y. Wang, K. Cunningham, P. Nagvajara, and J. Johnson, "Singular value decomposition hardware for mimo: State of the art and custom design," in *2010 International Conference on Reconfigurable Computing and FPGAs*, 2010, pp. 400–405.

[15] K.-J. Huang, W.-Y. Shih, J.-C. Liao, and W.-C. Fang, "A vlsi design of singular value decomposition processor used in real-time ica computation for multi-channel eeg system," in *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2013, pp. 413–416.

[16] N. D. Hemkumar and J. R. Cavallaro, "A systolic vlsi architecture for complex svd," *[Proceedings] 1992 IEEE International Symposium on Circuits and Systems*, vol. 3, pp. 1061–1064 vol.3, 1992. [Online]. Available: https://api.semanticscholar.org/CorpusID:123510010

[17] Y.-T. Hwang, W.-D. Chen, and C.-R. Hong, "A low complexity geometric mean decomposition computing scheme and its high throughput vlsi implementation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 4, pp. 1170–1182, 2014.

[18] X. Wang and J. Zambreno, "An fpga implementation of the hestenes-jacobi algorithm for singular value decomposition," *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, pp. 220–227, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:8041486

[19] A. Ibrahim, M. Valle, L. Noli, and H. Chible, "Assessment of fpga implementations of one sided jacobi algorithm for singular value decomposition," in *2015 IEEE Computer Society Annual Symposium on VLSI*, 2015, pp. 56–61.

[20] S. Majumder, A. Shaw, and S. Sarkar, "Hardware implementation of singular value decomposition," vol. 97, 2013, pp. 1–4.

[21] L. M. Ledesma-Carrillo, E. Cabal-Yepez, R. d. J. Romero-Troncoso, A. Garcia-Perez, R. A. Osornio-Rios, and T. D. Carozzi, "Reconfigurable fpga-based unit for singular value decomposition of large m x n matrices," in *2011 International Conference on Reconfigurable Computing and FPGAs*, 2011, pp. 345–350.

[22] Y. Ma and D. Wang, "Accelerating svd computation on fpgas for dsp systems," *2016 IEEE 13th International Conference on Signal Processing (ICSP)*, pp. 487–490, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:16491602

[23] B. Kabi, A. Routray, and R. Mohanty, "Fixed-point singular value decomposition algorithms with tight analytical bounds," 2015, pp. 12–18.

[24] U. Martinez-Corral, K. Basterretxea, and R. Finker, "Scalable parallel architecture for singular value decomposition of large matrices," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, 2014, pp. 1–4.

[25] L. Falaschetti, M. Alessandrini, G. Biagetti, P. Crippa, L. Manoni, and C. Turchetti, "Singular value decomposition in embedded systems based on arm cortex-m architecture," *Electronics*, vol. 10, pp. 6–7, 2020.

[26] D. Milford and M. Sandell, "Singular value decomposition using an array of cordic processors," *Signal Processing*, vol. 102, pp. 163–170, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016516841400125X

[27] J. Verth, *Mathematical Background*, 2010, pp. 3–28.

[28] S. Penev, “Linear causal modeling with structural equations by stanley mulaik,” *Australian New Zealand Journal of Statistics*, vol. 53, pp. 1–2, 2011.