ISTANBUL TECHNICAL UNIVERSITY
ELECTRICAL-ELECTRONICS FACULTY

AN FPGA IMPLEMENTATION OF DIGITAL ARCHITECTURE TO ESTIMATE THE EIGENVALUE OF ASYMMETRIC MATRIX

SENIOR DESIGN PROJECT

Elif ÖZTÜRK
İlayda KÖSEOĞLU

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

JUNE 2021

ISTANBUL TECHNICAL UNIVERSITY
ELECTRICAL-ELECTRONICS FACULTY

AN FPGA IMPLEMENTATION OF DIGITAL ARCHITECTURE TO ESTIMATE THE EIGENVALUE OF ASYMMETRIC MATRIX

SENIOR DESIGN PROJECT

Elif ÖZTÜRK
(040170208)

İlayda KÖSEOĞLU
(040180742)

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

Project Advisor: Prof. Dr. Müştak Erhan YALÇIN

JUNE 2021

İSTANBUL TEKNİK ÜNİVERSİTESİ
ELEKTRİK-ELEKTRONİK FAKÜLTESİ

ASİMETRİK MATRİSİN ÖZDEĞERİNİ HESAPLAYAN DİJİTAL MİMARİ FPGA UYGULAMASI

LİSANS BİTİRME TASARIM PROJESİ

Elif ÖZTÜRK
(040170208)

İlayda KÖSEOĞLU
(040180742)

Proje Danışmanı: Prof. Dr. Müştak Erhan YALÇIN

ELEKTRONİK VE HABERLEŞME MÜHENDİSLİĞİ BÖLÜMÜ

HAZİRAN, 2021

We are submitting the Senior Design Project Report entitled as “AN FPGA IMPLEMENTATION OF DIGITAL ARCHITECTURE TO ESTIMATE THE EIGENVALUE OF ASYMMETRIC MATRIX”. The Senior Design Project Report has been prepared as to fulfill the relevant regulations of the Electronics and Communication Engineering Department of Istanbul Technical University. We hereby confirm that we have realized all stages of the Senior Design Project work by ourselves and we have abided by the ethical rules with respect to academic and professional integrity .

Elif ÖZTÜRK .....
(040170208)

İlayda KÖSEOĞLU .....
(040180742)

FOREWORD

For As pre-graduates of Istanbul Technical University, we are very grateful for all the academic and social experiences our school offers us. We are proud to state that being a part of such a prestigious school has given us not only an innovative engineering understanding but also a respectful perspective on world values.

We would like to thank our dear Prof. Dr. Müştak Erhan YALÇIN for his support throughout the project and for deepening the study. We are grateful for his guidance in bringing our work to the academic world and providing us with such a unique experience. We would also like to thank our teacher's assistants for their help.

Special thanks to each other for working with enthusiasm despite the challenging conditions of the pandemic. Also, we wish to convey our love to our family, who has always supported us with love.

Finally, we commemorate the Başöğretmen Gazi Mustafa Kemal ATATÜRK, who led us on the way to modern civilization, with respect and gratitude, and present our deep respect and love.

June 2021

Elif ÖZTÜRK
İlayda KÖSEOĞLU

TABLE OF CONTENTS

Page

ABBREVIATIONS

BC **:** Boundary Cell
CGR **:** Column-wise Givens Rotation
CORDIC **:** Coordinate Rotation Digital Computer
DOA **:** Direction of Arrival
Eq. **:** Equation
FPGA **:** Field Programmable Gate Array
FSM **:** Finite State Machine
GG **:** Givens Generation
GR **:** Givens Rotation
IC **:** Internal Cell
IP **:** Intellectual Property
LUT **:** Look Up Table
MATLAB **:** Matrix Laboratory
MGS **:** Modified Gram Schmidt
MIMO **:** Multiple Input Multiple Output
MSB **:** Most Significant Bit
RU **:** Row Update
SA **:** Systolic Array
TSA **:** Triangular Systolic Array
QRD **:** QR Decompositon

SYMBOLS

$\in$	**:** Element of
λ	**:** Eigenvalue
T	**:** Transpose

LIST OF TABLES

Page

LIST OF FIGURES

Page

AN FPGA IMPLEMENTATION OF DIGITAL ARCHITECTURE TO ESTIMATE EIGENVALUES OF ASYMMETRIC MATRIX

SUMMARY

In this thesis, the digital circuit design that performs the eigenvalue calculation of asymmetric square matrices with real-valued elements is studied. For the eigenvalue calculation, the input matrix is factorized into orthogonal Q and upper triangular R matrix in the QR Decomposition block as a first step. Then, the RQ product is calculated in the Matrix Product block. Thus the updated matrix is obtained. It is checked in the Diagonal Control block whether this matrix is in the form of an upper diagonal. These processes are repeated until the upper diagonal matrix form is reached. The diagonals of the resulting matrix show the real-valued eigenvalues. These stages are referred to as QR Algorithm in the literature. However, when the matrix having complex eigenvalue pairs enters the system, the QR Algorithm becomes insufficient to create the upper triangular matrix form. The value in the bottom row of the diagonal element with these complex conjugate pairs is not zero. A 2x2 submatrix is drawn from the matrix having this nonzero element as the (2,1) value of the new matrix. Then this 2x2 dimensional matrix is given to the Solving Quadratic Equation block. In this block, the quadratic equation is obtained by taking the determinant and setting it to zero. By solving this equation, a complex conjugate eigenvalue pair is obtained. During the implementation of this system, square root and basic mathematic (addition, subtraction, multiplication, and division) operations were used. This system is implemented twice using two separate QR Decomposition blocks designed with the Modified Gram Schmidt (MGS) and the Givens Rotation (GR) algorithms. At first, both algorithms are implemented in MATLAB. Then error and parameter-based comparison analyzes are examined. In digital design, the parallelization feature of FPGA technology is utilized for a time-efficient system. Parallelization is managed by the Systolic Array (SA) architecture for both algorithms. This architecture consists of a combination of sub-modules that enable the concurrent execution of the MATLAB code loops. Triangular SA (TSA) architecture is created to obtain the upper triangular R matrix. This architecture has two different submodules as Boundary Cell (BC) and Internal Cell (IC). For an nxn-sized matrix, n BC modules and n(n-1)/2 IC modules are utilized. While the overall structure is similar, the internal cells of the modules differ in the MGS and GR algorithms. When the MGS algorithm is implemented, the designed BC module calculates the norm of the columns and obtains the diagonal values of the R matrix. Then, the columns of the Q matrix are obtained by dividing the column values by the obtained norm value. In the IC module, the upper diagonal values of the R matrix are calculated, and the input matrix is updated and given to the following line of the TSA architecture. The GR algorithm calculates the Givens Rotation matrix and multiplies it with the input matrix to obtain the R matrix. For the calculation of the Q matrix, the equation A=QR is used. After completing the digital

designs, MGS and GR algorithms are compared over parameters such as error amounts, area usage, and calculation time. The results obtained from the eigenvalue calculation design using the MGS algorithm have been submitted to the Cellular Nanoscale Networks and Applications (CNNA) 2021 Conference.

ASİMETRİK MATRİSİN ÖZDEĞERLERİNİ HESAPLAYAN DİJİTAL MİMARİ FPGA UYGULAMASI

ÖZET

Bu tezde tüm elemanları gerçek değerlerden oluşan asimetrik kare matrisler in özdeğer hesabı gerçekleyen devre tasarımı çalışılmıştır. Özdeğer hesabı için öncelikle giriş matrisi ortogonal Q ve üst üçgensel R matrisine QR Ayrıştırma bloğu içinde ayrıştırılmıştır. Daha sonra RQ çarpımı Matris Çarpımı bloğunda hesaplanarak güncellenmiş ara matris elde edilmiştir. Bu matrisin üst köşegen formunda olup olmadığı Köşegen Kontrol bloğunda kontrol edilmiştir. Matris üst köşegen formuna ulaşıncaya kadar yukarıda bahsedilen işlemler tekrarlanmıştır. Elde edilen matrisin köşegenleri gerçek değerli özdeğerleri gösterir. Bu aşamalar literatürde QR Algoritması olarak geçmektedir. Ancak karmaşık özdeğere çiftine sahip matris sisteme verildiğinde QR Algoritması üst üçgensel matris formu oluşturmakta yetersiz kalmaktadır. Bu karmaşık eşlenik çiftlerin bulunduğu köşegen elemanın alt satırındaki değer sıfırlanmaz. Bu sıfırlanmayan eleman yeni matrisin (2,1) değerini oluşturacak şekilde matristen 2x2 boyutlu alt matris çekilir. Ardından bu 2x2 boyutlu matris İkinci Dereceden Denklem Çözme bloğuna verilir. Bu blokta determinant alınıp sıfıra eşitlenerek ikinci dereceden denklem elde edilir. Bu denklem çözülerek karmaşık eşlenik özdeğer çifti elde edilir. Bu sistem gerçeklenirken sadece temel matematik (toplama, çıkarma, çarpma, bölme) ve karekök alma işlemleri kullanılmıştır. Bu sistem, farklı QR Ayrıştırma blokları tasarlanarak iki kere gerçeklenmiştir. İlk QR Ayrıştırma bloğu için Değiştirilmiş Gram Schmidt (MGS) yöntemi kullanılmıştır. İkinci blokta ise Givens Rotasyon (GR) yönteminden yararlanılmıştır. Her iki algoritma önce MATLAB üzerinde gerçeklenerek hata analizleri ve parametre bazlı karşılaştırmaları yapılmıştır. Bu algoritmaları daha hızlı çalıştırabilmek için FPGA teknolojisinin paralelleştirme özelliği kullanılmıştır. Paralelleştirme, her iki algoritma için de Sistolik Dizi (SA) mimarisine uygun olarak yapılmıştır. Bu mimari, MATLAB kodunda yer alan döngülerin aynı anda gerçeklenmesini sağlayan alt modüllerin uygun şekilde birleştirilmesiyle oluşur. Üst üçgensel R matrisi elde edilebilmesi için Üçgensel SA (TSA) mimarisi oluşturulmuştur. Bu mimari Köşe Hücre (BC) ve İç Hücre (IC) olarak iki farklı alt modüle sahiptir. nxn boyutlu matris için BC modülü n, IC modülü ise n(n-1)/2 defa kullanılır. Genel yapı benzer olmakla birlikte, modüllerin iç yapıları MGS ve GR algoritmalarında farklılık gösterir. MGS algoritması gerçeklenirken tasarlanan BC modülü sütunların norm hesabını yaparak R matrisinin köşegen değerlerini elde eder. Daha sonra sütun elde edilen norm değerine bölünerek Q matrisin sütunları elde edilir. IC modülünde ise R matrisin üst köşegensel değerleri hesaplanır ve giriş matrisi güncellenerek TSA mimarisinin bir sonraki satırına verilir. GR algoritması ise R matrisini elde etmek için Givens Rotasyon matrisi hesaplayıp giriş matrisi ile çarpar. Q matrisinin hesabı için ise A=QR eşitliğinden yararlanılır. Sayısal tasarım tamamlandıktan sonra hata miktarları, alan kullanımları ve hesaplama

süresi gib parametreler üzerinden MGS ve GR algoritmaları karşılaştırılmıştır. MGS algoritması kullanılarak gerçekleştirilen özdeğer hesabı tasarımından elde edilen sonuçlar Hücresel Nano Ölçekli Ağlar ve Uygulamaları (CNNA) 2021 Konferansına sunulmak üzere gönderilmiştir.

1. INTRODUCTION

1.1 Purpose of Project

Eigenvalue calculation is essential in the communication technologies and digital signal processing fields such as MIMO detection for wireless communication systems, adaptive beamforming, estimating the Direction of Arrival (DOA), etc. Considering communication technologies, speed is one of the main concerns in these fields. However, the eigenvalue calculations are handled through iterative methods causing excessive time loss. It is aimed to utilize Systolic Array architecture to overcome the time loss by using the parallelization feature of FPGA technology.

1.2 Literature Review

For eigenvalue estimation, there exist two popular algorithms coherent with SA architecture: Jacobi Theorem [1, 2] and QR Algorithm [3]. Both algorithms use an iterative approach to eliminate the lower-diagonal elements of the matrix to find the eigenvalues. However, Jacobi Theorem becomes insufficient for the eigenvalue calculations of asymmetrical matrices. Therefore, QR Algorithm is studied in this project.

QR Decomposition (QRD) forms the basis of the QR Algorithm, factoring matrix A into a product A = QR of an orthogonal matrix Q and upper triangular matrix R. Once Q and R matrices are obtained, they are multiplied in reverse order to calculate the input of the QRD module of the next iteration.

There are two basic decomposition algorithms: Givens Rotation (GR) and Modified Gram Schmidt (MGS) Orthogonalization [4, 5]. In the digital design of GR, Coordinate Rotation Digital Computer (CORDIC) cores are used in several different studies [6, 7]. However, this contradicts the purpose of this study, causing excessive delays in the design. Alternatively, design can be implemented by using only basic

mathematical modules and square root modules [8]. Since a faster working system was aimed, MGS and GR implemented with the alternative method.

1.2.1 Analyzation of Jacobi eigenvalue theorem

Jacobi Eigenvalue algorithm is a method used in diagonalization to calculate the eigenvalues of real symmetric matrices. In this method, rotations are performed iteratively until eliminating all off-diagonal elements.

First of all, matrix entry is made in the "data load" section. Then rotation matrix is calculated with rotation angle. Matrix multiplication is performed using the obtained R matrix.

In the data exchange part;

- First odd column does not change.
- Last odd column is moved to the next column.
- First even column is moved to the next column.
- Other odd columns are moved to next odd column.
- Other even columns are moved to the previous odd column.

The same column exchange steps are repeated for row exchange [1].

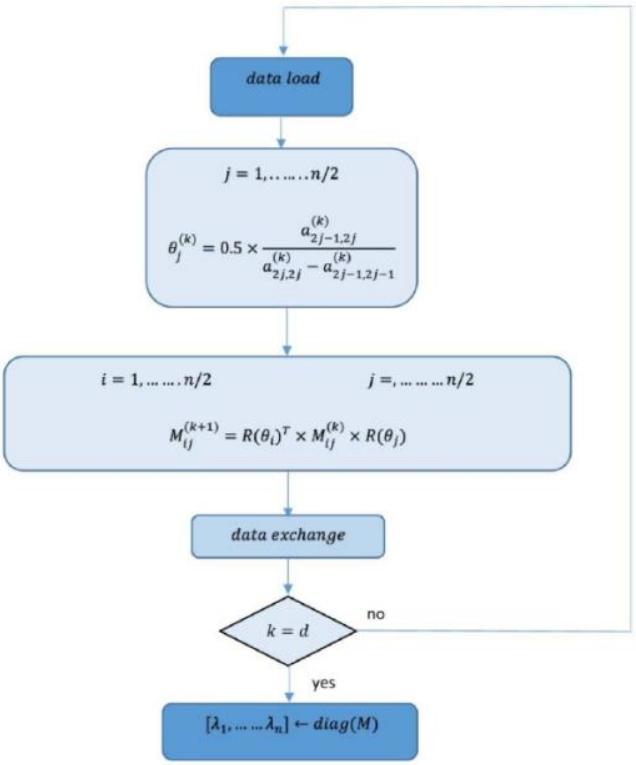


Figure 1.2.1: Jacobi eigenvalue algorithm [2].

1.2.2 Analyzation of QR algorithm

QR algorithm calculates eigenvalues through the factorization of Q orthogonal and R upper triangular matrices with QR decomposition. The RQ multiplication forms the input of the QRD at the next iteration (Algorithm 1). This process continues until the lower triangular elements of input A(n,n) $\in$ IR matrix approaches zero. As a result, eigenvalues are obtained from the diagonal elements. However, if A is an asymmetric matrix, complex conjugate eigenvalue pairs might exist. In such a case, the QR algorithm becomes insufficient to calculate these pairs. To overcome this, one element below each diagonal is checked. When a non-zero element is observed, it indicates a complex conjugate eigenvalue pair. To find this pair, a 2x2 matrix containing this element at position (2,1) is taken. In other words, the diagonal element at one row above the mentioned non-zero element forms the position (1,1) whereas one column right of this element forms the position (2,2) of the 2x2 matrix. This new matrix is used to calculate these complex conjugate eigenvalues via taking the determinant and then solving the quadratic equation.

Algorithm 1: QR algorithm [3]

input : Matrix A
output : Eigenvalues = Diagonal elements of A^{out}
1: **while** ~ *Lower Diagonal elements of* A^i *approach to zero* **do**
2: [Q, R] ← QR Decomposition(A^i);
3: i ← i+1;
4: A^i ← R*Q;
5: **end while**

1.2.2.1 Givens rotation principle

Givens Rotation principle that forms a plane rotation spanned by two coordinate axes. With this principle, the lower triangular part of the original matrix is brought closer to zero at each iteration. Its process can be explained in six steps [8].

Step 1: Starting from the first column of an n x n asymmetric matrix, the first non-zero lower triangular element is determined to make that element zero. For this, a n x n Givens matrix should be formed with subsection in which its -sin θ element is placed at the same row and column number of previously determined element. Similarly cos θ is placed as the first diagonal element of the column. The other diagonals of Givens are always defined as one and all other elements become zero. Then this G matrix is

multiplied with the original matrix form left. The resultant matrix will have a zero element in the previously determined spot. This procedure is replied until all the first column is checked by using the latest resultant matrix instead of the original one at each step.

Step 2: Step 1 is repeated for all other columns until the whole matrix is checked.

Step 3: The last obtained matrix becomes the R of the decomposition. To find Q all G matrices determined are multiplied respectively from left. Then, the transpose of this multiplication is calculated.

Step 4: To calculate the transpose, all diagonal elements of the matrix is kept and the other elements are changed in pairs by switching each element's row and column values. By this way, Q of the decomposition is achieved. This matrix is called the plane rotator acting in the plane of row and column values determined in step 1.

Step 5: A new matrix is calculated with R*Q. Then these steps are repeated with this new matrix until the lower triangular of the new matrix becomes all zero.

Step 6: When step 5 is completed and a new upper triangular matrix is obtained, its diagonal elements indicate the eigenvalues of the original matrix.

$$
\begin{aligned}
&\boldsymbol{Q} = \boldsymbol{I}_m \\
&\text{for } i = 1 : n-1 \\
&\quad \text{for } k = i+1 : m \\
&\quad\quad c = \boldsymbol{A}(i,i)/\sqrt{\boldsymbol{A}(k,i)^2 + \boldsymbol{A}(i,i)^2} \\
&\quad\quad s = \boldsymbol{A}(k,i)/\sqrt{\boldsymbol{A}(k,i)^2 + \boldsymbol{A}(i,i)^2} \\
&\quad\quad \boldsymbol{A}([i,k], i:n) = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \boldsymbol{A}([i,k], i:n) \\
&\quad\quad \boldsymbol{Q}([i,k], 1:n) = \begin{bmatrix} c & -s \\ s & c \end{bmatrix}^T \boldsymbol{Q}([i,k], 1:n) \\
&\quad \text{end} \\
&\text{end} \\
&\boldsymbol{R} = \boldsymbol{A}
\end{aligned}
$$

Figure 1.2.2.1: Givens rotation algorithm [9].

1.2.2.2 Gram Schmidt theorem

Gram Schmidt process is a method used to orthonormalize any vector sets existing in inner product space also known as Rn. The QR decomposition is achieved by applying the Modified Gram–Schmidt process to the column vectors of a full column rank matrix. With this composition, a triangle and an orthogonal matrix are obtained.

The MGS Orthogonalization [4] is used to form an orthogonal basis by creating Q(m,n)∈IR orthogonal and R(n,n)∈IR upper triangular matrices from a non-orthogonal basis of A(m,n)∈IR (where m ≥ n) (Algorithm 2). As a first step, the algorithm separates the A matrix into n columns such as

$$A = [a_1|a_2|\dots|a_n], \qquad (1 \le i \le n). \qquad (1.2.2.2.a)$$

The diagonal values (r_{ii}) of the upper triangular R matrix are found by taking the norm of this column vector where a_i is the i_{th} column and a^i is the i^{th} iteration of A

$$r_{ii} = \|a_i^i\|. \qquad (1.2.2.2.b)$$

Similarly, the Orthonormal Q column (q_i) is obtained by dividing this column vector by its norm value

$$q_i = \frac{a_i^i}{r_{ii}}, \qquad (1 \le i \le m). \qquad (1.2.2.2.c)$$

The off-diagonal elements (r_{ij}) of the upper triangular R matrix are the inner product (×) of the transpose of orthonormal q vector and the column vector of the input matrix such as

$$r_{ij} = q_i^T \times a_j^i, \qquad (i+1 \le j \le m). \qquad (1.2.2.2.d)$$

In the calculation of each lower diagonal element, this product value becomes zero, so the upper triangular matrix is obtained. The column vector (a_j^{i+1}) is updated by subtracting the projection of each column vector (a_j^i) from the original column vectors at each iteration such as

$$\begin{aligned} a_j^{i+1} &= a_j^i - r_{ij} q_i \\ &= a_j^i - \frac{(a_i^i)^T a_j^i a_i^i}{r_{ii}^2}, \\ (2 \le i+1 \le m), &\qquad (i+1 \le j \le n). \end{aligned} \qquad (1.2.2.2.e)$$

Algorithm 2: Modified Gram Schmidt Algorithm [5]

input : Matrix A
output : Upper Diagonal R and Orthogonal Q Matrices

```
1: for j = 1:n do
2:     v ← A( : , j );
3:     for i = 1 : j – 1 do
4:         R(i , j) ← Q(: , i)' * A(: , j);
5:         v ← v – R(i , j) Q(: , i);
6:     end for
7:     R(j , j) ← norm(v);
8:     Q(: , j) ← v / R(j , j);
9: end for
```

1.2.3 Coordinate rotation digital computer (CORDIC) algortihm

The description of CORDIC arithmetic was presented by Jack E. Volder in 1959. It can be defined as a shift and addition algorithm. With this algorithm, recursive most of the basic functions can be calculated. For this, xed point arithmetic operations are utilized.

CORDIC is widely used in many applications due to its cost-effectiveness in hardware implementations compared to those using multiplier and lookup tables. In addition, the numerical stability of the algorithm also becomes very important for xed-point applications. Unfortunately, due to the iterative methods utilized in the CORDIC algorithm, data-dependent latency occurs [6, 7, 9].

$[\, x', y'] = \text{CORDIC\_rotation\_mode}(x,\, y,\, \theta,\, N,\, K)$
$\sigma_{-1} = sign(\theta)$
$x_0 = -\sigma_{-1} \cdot y$
$y_0 = \sigma_{-1} \cdot x$
$\theta_0 = \theta - \sigma_{-1} \cdot (\pi/2)$
for $i = 0 : N$
  $\sigma_i = sign(\theta)$
  $x_{i+1} = x_i - \sigma_i \cdot 2^{-i} \cdot y_i$
  $y_{i+1} = y_i + \sigma_i \cdot 2^{-i} \cdot x_i$
  $\theta_{i+1} = \theta_i - \sigma_i \cdot \arctan(2^{-i})$
end
$x' = x_N \cdot K$
$y' = y_N \cdot K$

(a)

$[\, z, \theta\,] = \text{CORDIC\_vectoring\_mode}(x,\, y,\, N,\, K)$
$\sigma_{-1} = -sign(y)$
$x_0 = -\sigma_{-1} \cdot y$
$y_0 = \sigma_{-1} \cdot x$
$\theta_0 = \sigma_{-1} \cdot (\pi/2)$
for $i = 0 : N$
  $\sigma_i = -sign(y_0)$
  $x_{i+1} = x_i - \sigma_i \cdot 2^{-i} \cdot y_i$
  $y_{i+1} = y_i + \sigma_i \cdot 2^{-i} \cdot x_i$
  $\theta_{i+1} = \theta_i - \sigma_i \cdot \arctan(2^{-i})$
end
$z = x_N \cdot K$
$\theta = \theta_N$

(b)

Figure 1.2.3: a) CORDIC Rotation Mode Algorithm b) CORDIC Vectoring Mode Algorithm. [9]

1.2.4 Systolic array (SA) architecture

Systolic Array (SA) architecture is a network of processing element that enables the parallelization of the system. Through this architecture, the computations are concurrently handled thus providing time efficiency. As a trade-off, SA architecture consumes a wider area due to high degrees of pipelining. However, it provides a simple and straightforward design by utilizing the same modules multiple times. With this procedure, I/O requirements remain unchanged [10].

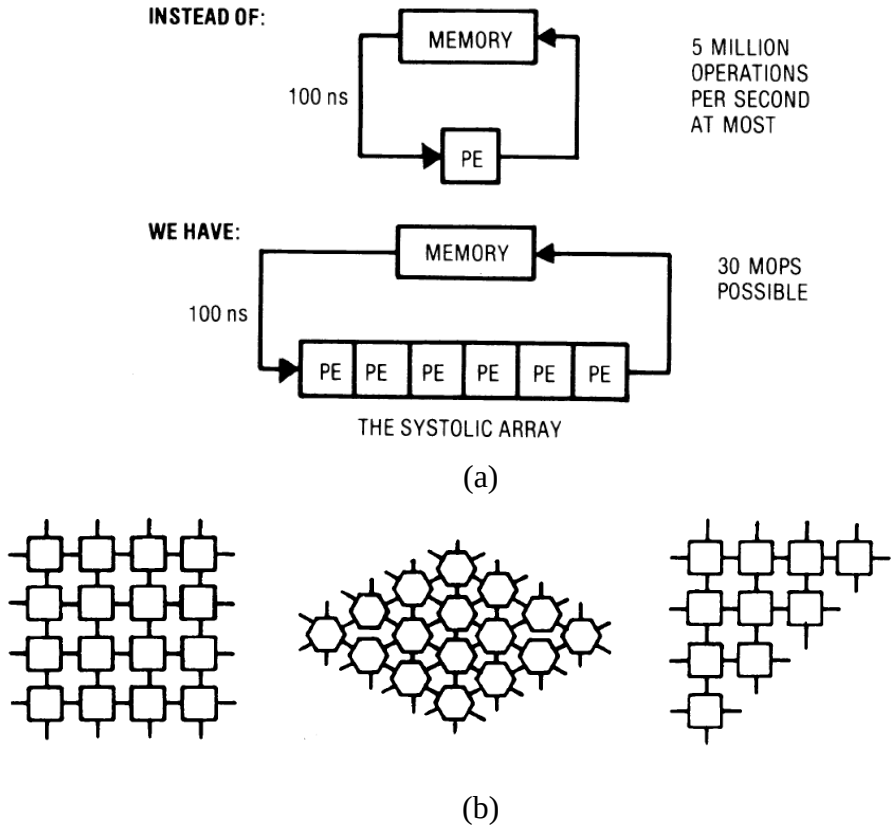


Figure 1.2.4 : a) Basic principle of systolic array. b) Two diamensional systolic arrays respectively R type, H type, T type. [10]

2. HIGH LEVEL IMPLEMENTATION WITH MATLAB

2.1 Objectives

When the implementation of a new algorithm is required, it is performed via MATLAB special functions at first. In this way, errors can be detected much faster and easier saving time and effort for digital design procedures. If accomplished, the used functions are rewritten in an algorithmic form to be directly transferred to digital design.

2.2 QR Algorithm

The QR algorithm mentioned in the literature review section (1.2.2) is implemented on MATLAB using Algorithm 1.

2.2.1 Iteration and resolution

QR algorithm is basically an algorithm based on iteration. After obtaining the Q and R matrices, matrix multiplication is made and this iteration is repeated until the lower triangular part becomes zero.

```
M=A;

for i = 1:1500

    [Q,R] = qr(M);  % q and r matrices calculation

    M=R*Q;          % matrix multipication for iteration

end
```

The iteration number is determined by an additional comparison block that is repeated until all lower triangular values become less than absolute tolerance. Thus, unnecessary calculations are avoided.

```
while ~exitFLAG
  -----
 %QR calculation
  -----
  M=R*Q;
   inc=0;
   for j=1:n
      for i=(j+2):n                % 2 rows under each diagonal
         if ( abs(M(i,j))<tol )    % compairing with tolerance
            inc=inc+1;             % if less enough increment inc value
         end
         i=i+1;
      end
      j=j+1;
   end
   if( inc==(((n-2)*(n-1)))/2 )  % Inc value must be as much as the sum of the
      exitFLAG=1;                % numbers from (n-2) to 0 to set flag and exit while.
   end
end
```

Due to the possibility of one or more conjugate pair existence explained in the following section, the lower triangular matrix might have non-zero values at 1 row under each diagonal element. These values refer to conjugate eigenvalue pairs that should be analyzed separately. For this reason, the comparison to zero starts with 2 rows under each diagonal, resulting in a total of ((n-2)(n-1))/2 computations.

2.2.2 Recalculation for conjugate eigenvalues pair

Asymmetric matrices differ from symmetric ones in that they have complex eigenvalues. If a matrix is made up of real values, its complex eigenvalues are conjugate pairs. Unfortunately, the QR algorithm becomes insufficient in converting these values to zero. To overcome this problem, these values are taken with their upper values to form 2 by 2 matrices so that their eigenvalues can be calculated quadratically.

The eigenvalue calculation for the 2 by 2 matrix A is done as follows:

- Subtract matrix A from the eigenvector.
- The determinant is calculated.
- The solution of the quadratic equation resulting from the determinant gives the eigenvalue value.

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

$$\det\begin{bmatrix} \lambda - a & -b \\ -c & \lambda - d \end{bmatrix} = \lambda^2 - (a+d)\,\lambda + (ad - bc) = \lambda^2 - \lambda\,\mathrm{tr}(A) + \det(A)$$

$$\lambda = \frac{\mathrm{tr}(A) \pm \sqrt{\mathrm{tr}^2(A) - 4\det(A)}}{2}$$

Figure 2.2.2: Eigenvalue calculation steps for 2 by 2 A matrix.

```
for i = 2 : n
   for j = i-1                    % 1 row under each diagonal
      if (abs(M(i,j)) > tol)      % compairing with tolerance
         B(1,1) = M(i-1,j);     % Determine the 2x2 matrix with the non-0 element
         B(2,1) = M(i,j);       % in the lower left corner.
         B(1,2) = M(i-1,j+1);
         B(2,2) = M(i,j+1) ;
         a = 1;
         b = -(B(1,1)+B(2,2));
         c = B(1,1)*B(2,2) - B(1,2)*B(2,1) ;
         delta=b*b-4*a*c;
         x1=(-b+(delta)^0.5)/(2*a);
         x2=(-b-delta^0.5)/(2*a);
         M(i,j) = 0;
         M(i-1,j) = x1;
         M(i,j+1) = x2;
      end
   end
end
```

2.3 Q and R Matrices Estimation

In order not to use the MATLAB “qr” function, the algorithms that obtain the Q and R matrices will be examined under this section.

2.3.1 Givens rotation algorithm for Q and R estimation

The MATLAB code below is implemented according to the algorithm given in Figure 1.2.2.1 in the subtitle 1.2.2.1 of the literature review section.

```
for j=1:n                                   % jth column
   for i=1:n                                % ith row
      if i>j
         if R(i,j)~=0
         Mnew=R; Gnew=G; Qnew=Q; % Computing the rotation matrix (Gnew)---
         c_cos= R(j,j)/(sqrt( (R(j,j)*R(j,j)) +  (R(i,j)*R(i,j)) ));
         s_sin= R(i,j)/(sqrt( (R(j,j)*R(j,j)) +  (R(i,j)*R(i,j)) ));
         Gnew(j,j)=c_cos;    Gnew(i,i)=c_cos;
         Gnew(j,i)=s_sin;     Gnew(i,j)= -s_sin;  %----------------------------------------
         R=Gnew*Mnew;                 % Matrix multipication for R matrix
         Q=Gnew*Qnew;                 % Matrix multipication for Q matrix
         end
      end
   end
end
```

2.3.2 Gram Schmidt algorithm for Q and R estimation

The MATLAB code given in Algorithm 2 in the subtitle 1.2.2.2 of the literature review section for Gram Schmidt is implemented directly. There is no need for any additions.

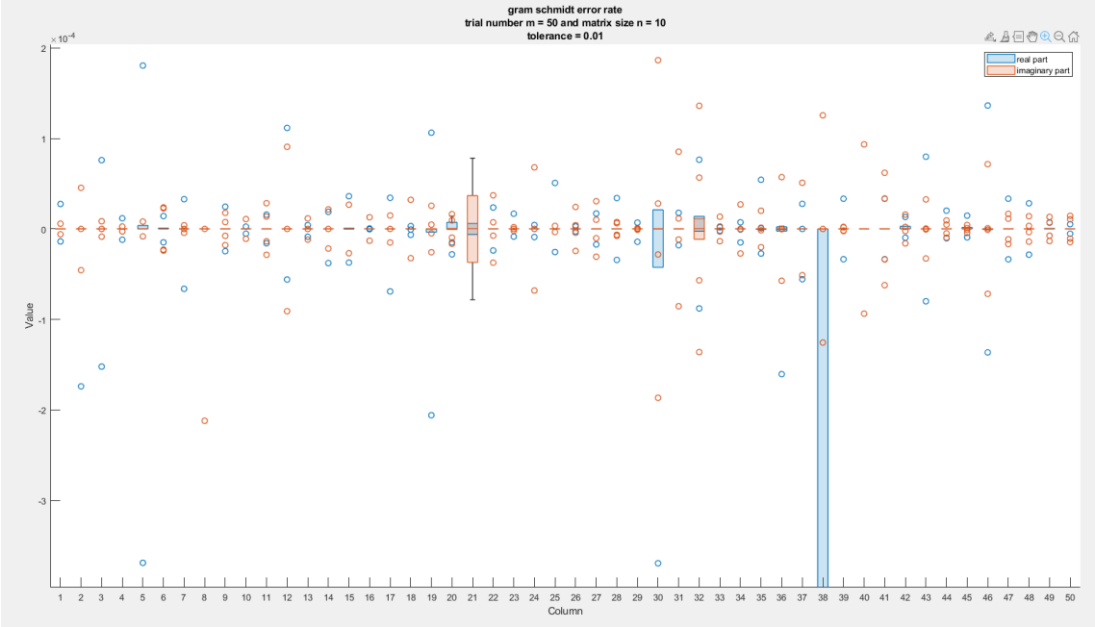
2.4 Error Rate Estimation and Comparison

In this section, error rates and comparisons of MATLAB results are given. At each comparison, only one variable is changed to observe its effect on the results. All matrices under this section are produced with the MATLAB's sprecial "randn (n)" function.

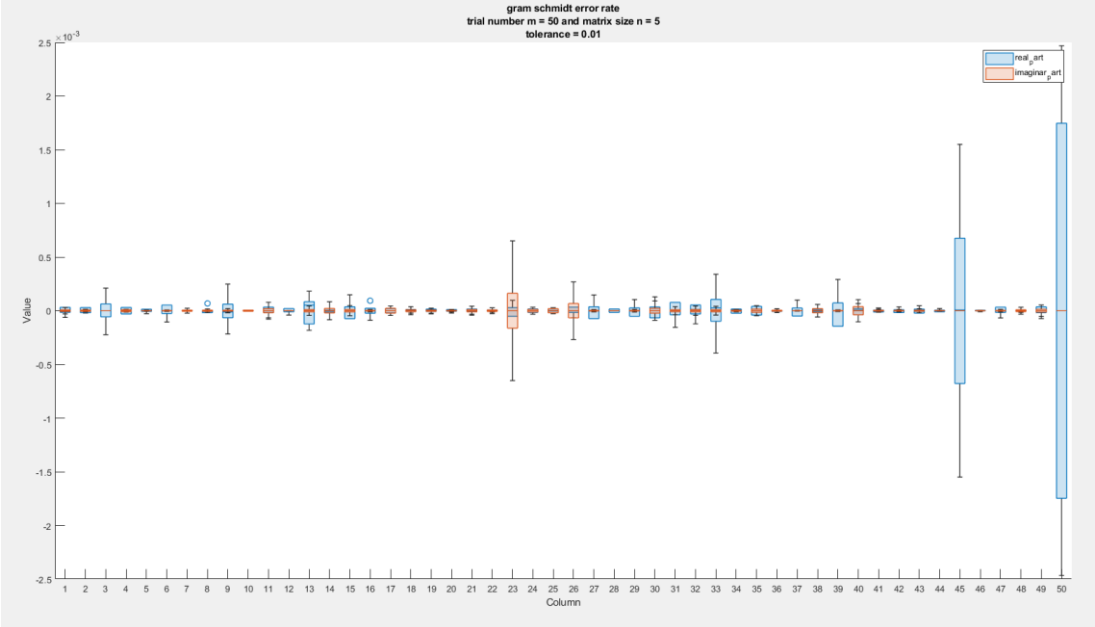
Error rate estimation was made according to the "$\lambda_{MATLAB} - \lambda_{GS \text{ or } GR}$" equation. λ_{GS} stands for eigenvalues of calculated by Gram Schmidt. Similarly λ_{GR} stands for eigenvalues of calculated by Givens Rotation.

2.4.1 Matrix size comparison

As can be seen in Figure 2.2.3.1, first for a 5x5 matrix and then for a 10x10 matrix calculations are presented. No significant change in the error rate according to the matrix size is observed.



(a)

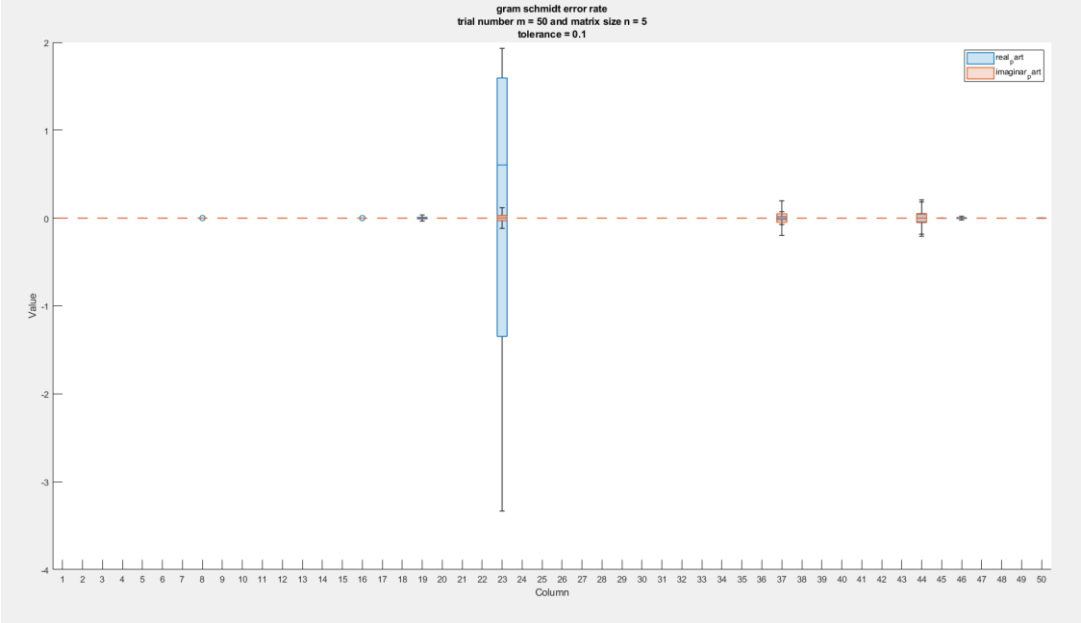


(b)

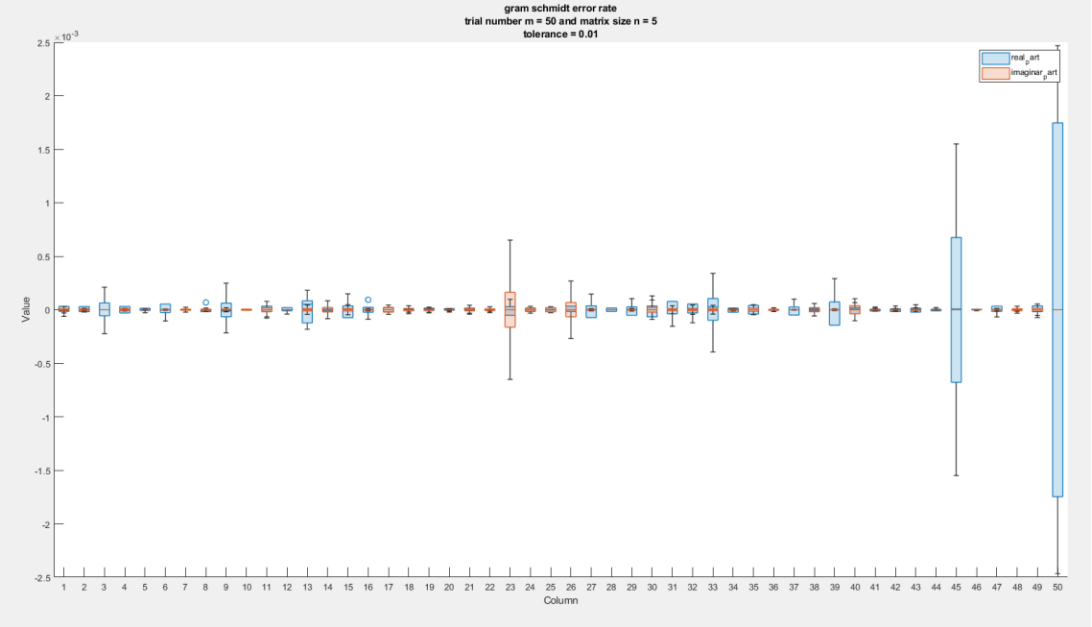
Figure 2.4.1: Matrix comparison for $\lambda_{MATLAB} - \lambda_{GS}$. a) 10x10 matrix b) 5x5 matrix

2.4.2 Tolerance for complex conjugate pairs comparison

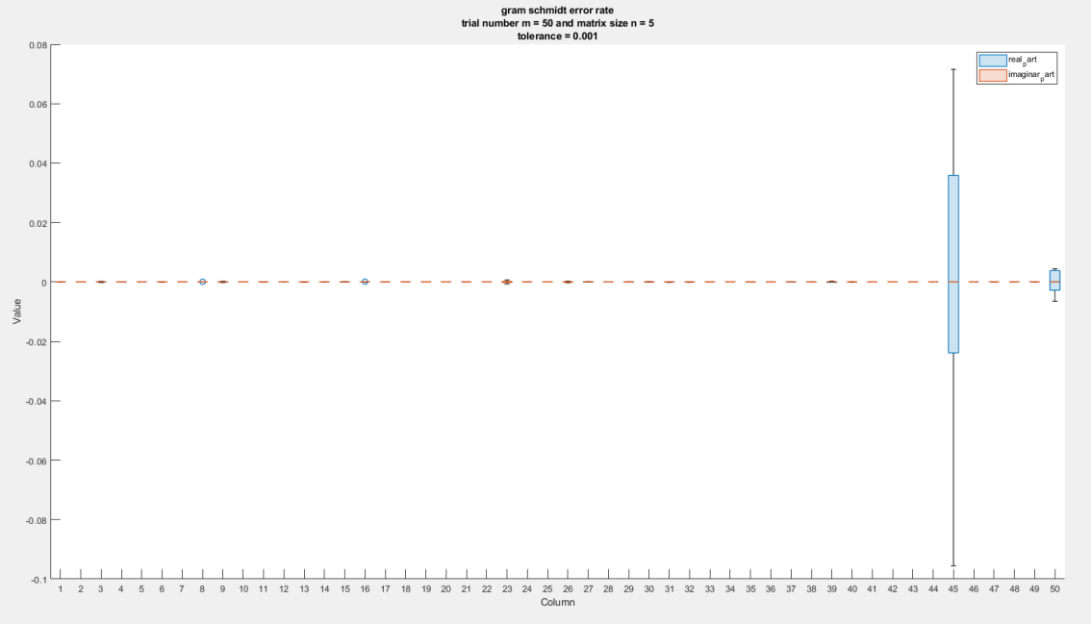
Most of the errors are observed due to complex conjugate pairs. It is expected for the bottom row of diagonal lines to become zero. However, the algorithms fail to do so when a complex conjugate eigenvalue pair occurs. To overcome this, 2 different tolerance values (10^{-5} and 10^{-2}) are defined. One of them determines how far the 2 sub-lines of the diagonal will approach zero, while the other controls the rows in the bottom non-zero rows of the diagonal. If this tolerance value is chosen too low (0.001), the value at the bottom of the diagonal is determined higher. Therefore, the algorithm assumes that value to refer to a complex conjugate eigenvalue despite approaching zero. As a result, extra unnecessary calculations takes place and a higher error rate is observed. Else if the number is too large (0.1), complex pairs calculation is skipped. As can be noticed in Figure 2.2.3.2, the most optimal tolerance parameter is set at 0.01. However, errors can still occur for the same reasons.



(a)



(b)



(c)

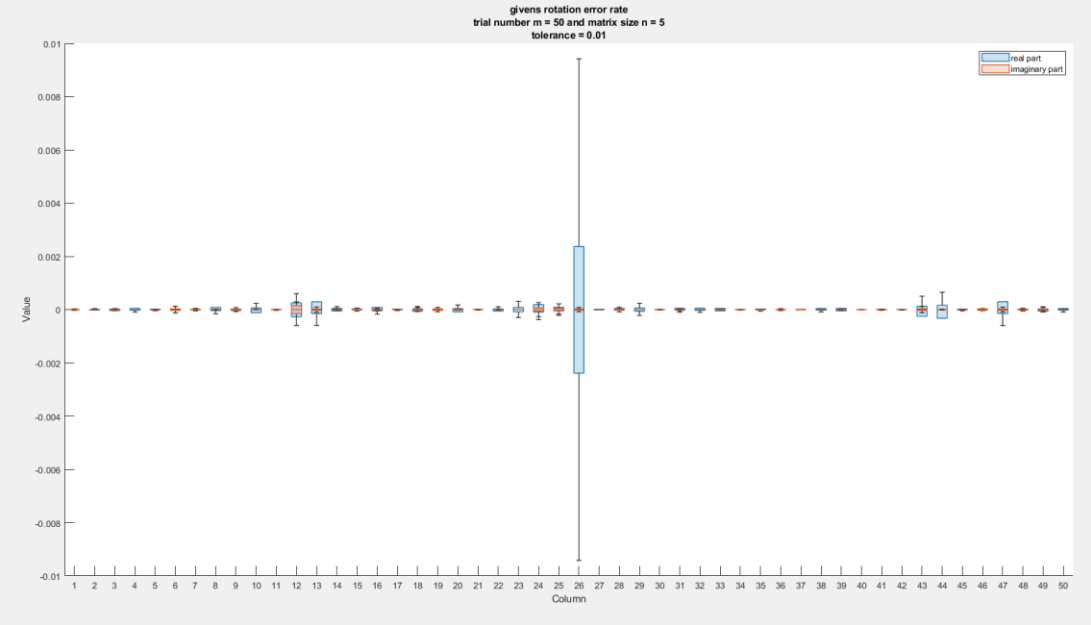
Figure 2.4.2: Tolerance for complex conjugate pairs comparison for λ_{MATLAB} - λ_{GS}.

a) Tolerance parameter: 0.1 b) Tolerance parameter: 0.01

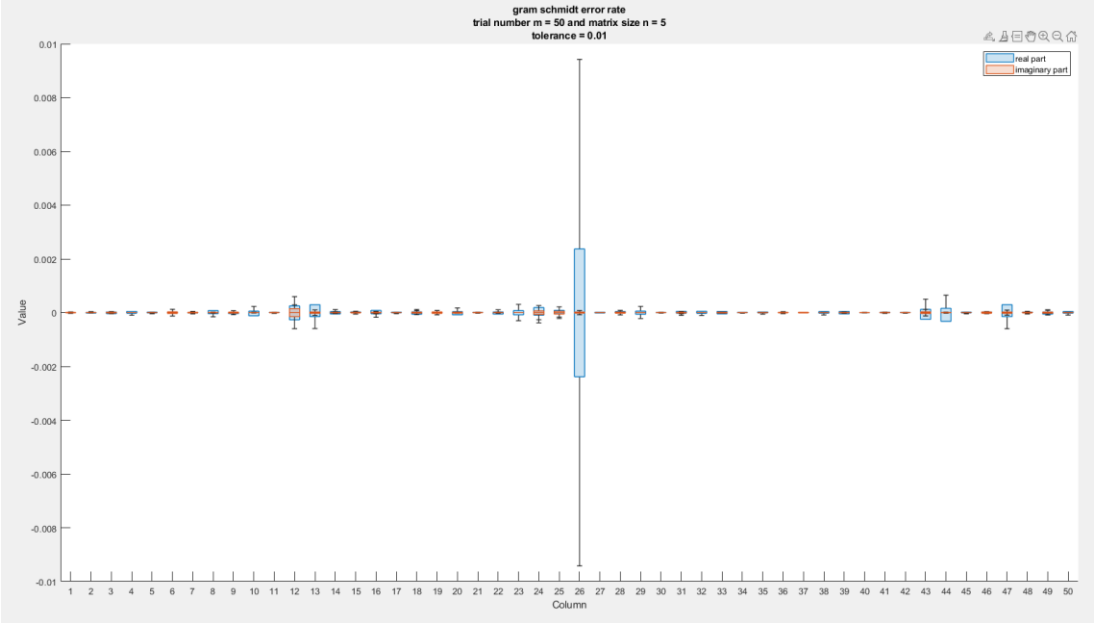
c) Tolerance parameter: 0.001

2.4.3 Algorithm comparison

Givens Rotation and Gram Schmidt algorithms are tested with the same random matrices. Both results are observed as identical. When the Q and R values are examined, it is noticed that they also are exactly the same.



(a)



(b)

Figure:2.4.3: Algorithm comparison. a) Gram schmidt: $\lambda_{MATLAB} - \lambda_{GS}$ b) Givens rotation: $\lambda_{MATLAB} - \lambda_{GR}$

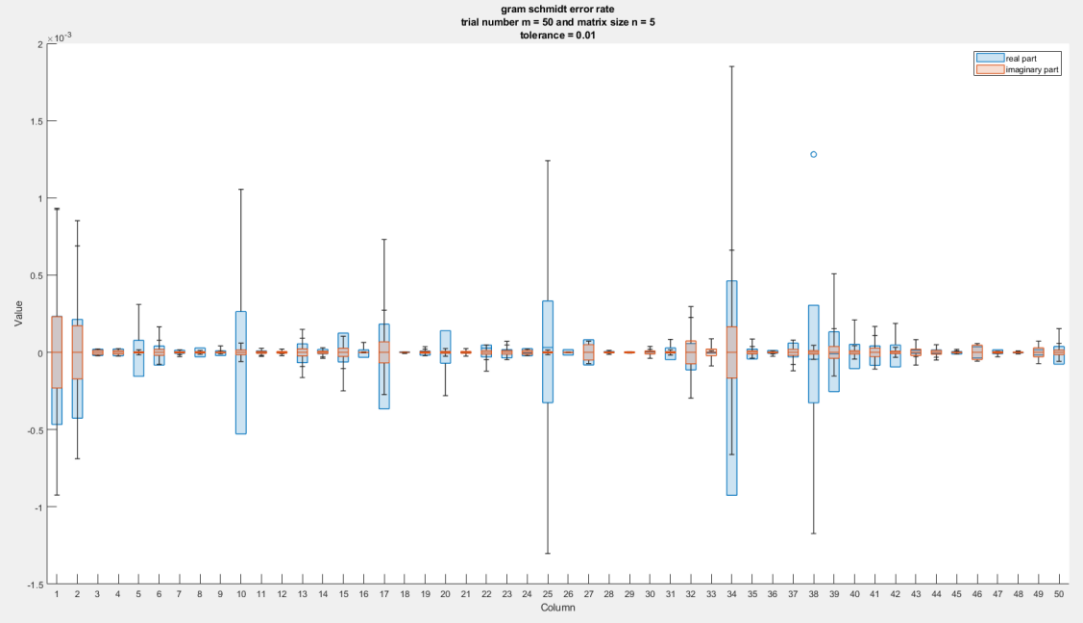
As can be observed in Fig 2.2.3.3, Gram Schmidt and Givens Rotation algorithms yield the same error rates. For this reason, their comparison of operation complexity is required In this comparison shown in Table 2.2.3.3, only the calculation of Q and R matrices for an n by n matrix is taken into consideration. Using the table, it can be concluded that Gram Schmidt has less operation complexity.

Table 2.4.3 : Operation comparision of algorithms implemented on MATLAB.

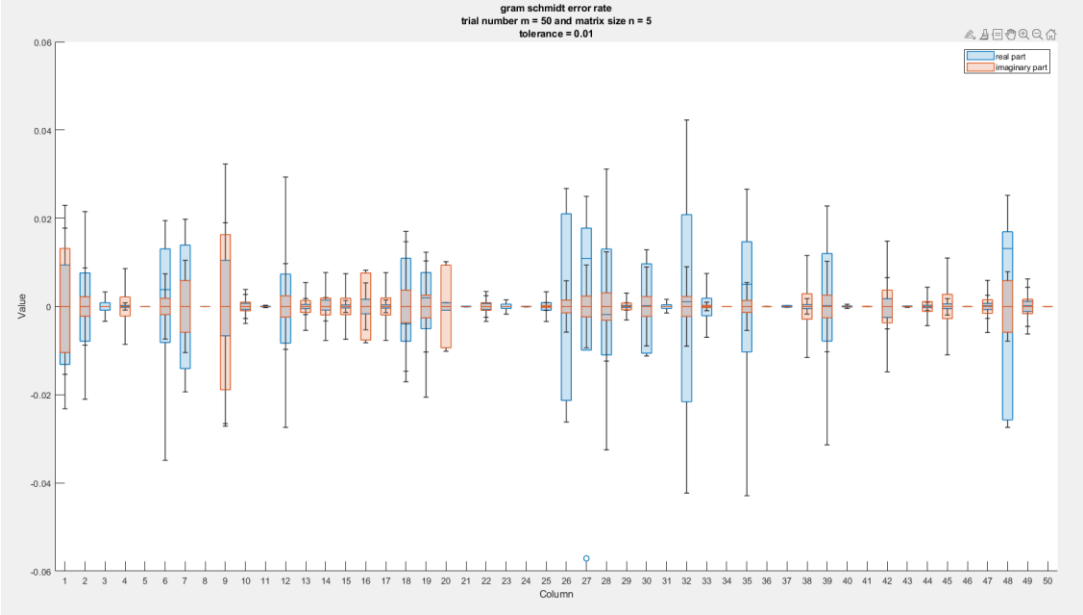
Operation	Givens Rotation	Gram Schmidt	Differences (Givens Rotation – Gram Schmidt)
Multiplication	$3n^2-3n$	$2n^2-n$	n^2-2n
Addition	n^2-n	n^2-n	-
Subtraction	-	$(n^2-n)/2$	$-(n^2-n)/2$
Division	n^2-n	n	n^2
Square Root	n^2-n	n	n^2

2.4.4 Size comparison for matrix elements

Error rates decreases when the size of the produced matrix elements is multiplied by 100. However, it increases when the size is divided by 100. The main reason for this is that as the matrix elements values increase, the amount of iteration also increases.



(a)



(b)

Figure 2.4.4: Size comparison for matrix elements for λ_{MATLAB} - λ_{GS}.

a) multiplied by 100 b) divided by 100

3. LOW LEVEL IMPLEMENTATION

3.1 Objectives

Algorithms that are run correctly in MATLAB will be implemented on FPGA at this stage. It is aimed to produce faster results by taking advantage of the parallelization feature of the digital design.

In this study, matrice elements are limited between -2 and +2. When converting to binary form, 16 bits are required in total. The MSB is assigned as the sign bit. The next 5 bits are utilized as the decimal part and the rest as the fraction part.

3.2 Square Root Algorithm and Its Digital Hardware Design

The square root process is frequently used in eigenvalue estimation algorithms. To accomplish this, either CORDIC IP or a separate algorithm can be utilized. However, since the input data of CORDIC IP is limited between -1 and +1, it is not suitable for this study. For this reason, the calculation is handled by a suitable algorithm [11].

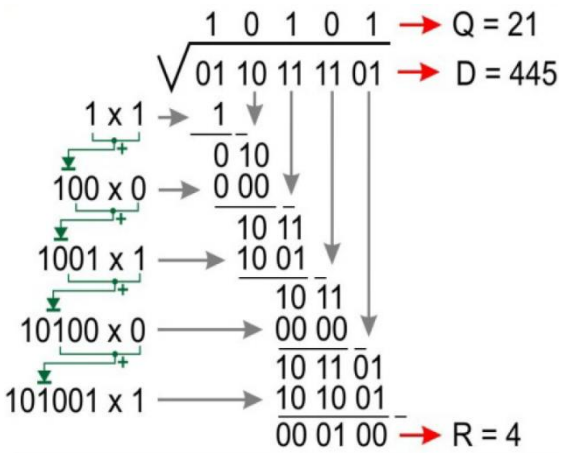


Figure 3.2 : a) Square root algorithm example. [11]

This algorithm is used for the square root operation in the BC module. Input data Radicand (D) register, other Remainder (R), Quotient (Q), Partial factor (F) registers are assigned to zero. If the input D is an odd-bit number, it is expanded with '0' as its MSB. The digits of D are grouped in pairs starting from LSB. For each group, one iteration is operated as depicted in Algorithm 3.

Algorithm 3: One Iteration of Square Root Algorithm [11]

input : 2-bits of D
output : Q^{i+1}, F^{i+1}, R^{i+1}
1: Partial remainder ← 2-bit D;
2: Partial factor ← ((F^i << 1) | 1);
3: **if** *Partial remainder* ≥ *Partial factor* **then**
4: Q^{i+1} ← (Q_i << 1) | 1;
5: F^{i+1} ← ((F^i + F^i [0]) << 1) | 1;
6: **else**
7: Q^{i+1} ← (Q^i << 1) | 0;
8: F^{i+1} ← ((F^i + F^i [0]) << 1) | 0;
9: **end if**
10: R^{i+1} ← ((R^i – (F^i F^i [0])) << 2) | D;

The iteration (Algorithm 3) starts from the comparison and is repeated until the final square root is achieved. When the process ends, the resultant Q is assigned as the square root and R as the remainder.

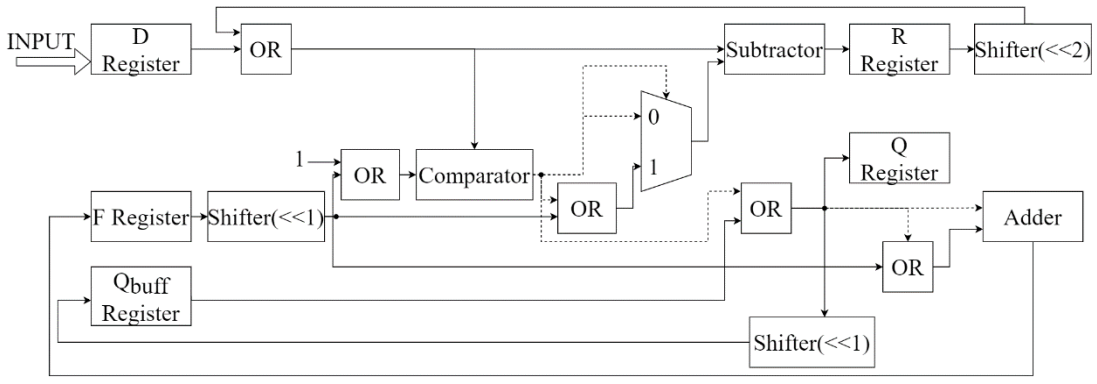


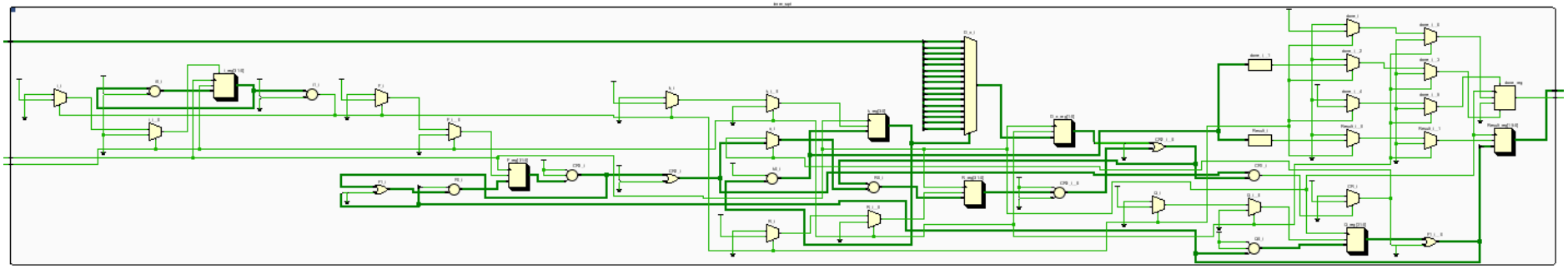
Figure 3.2 : b) Square root algorithm circuit. [11]

Radicant (D) input goes into iteration in bits of 2. In order to give these bits sequentially, multiplexer is used in addition to the circuit diagram in Figure 3.2 b. By increasing the selection bit, the D input bits are received in pairs every iteration. In the end, (bit number of D) / 2 times iteration is completed. Each iteration lasts 2 clock cycles. In order to avoid timing errors, a 2-clock loop was added as a custom made delay unit, and the other iteration was passed at the end of the loop. Therefore,

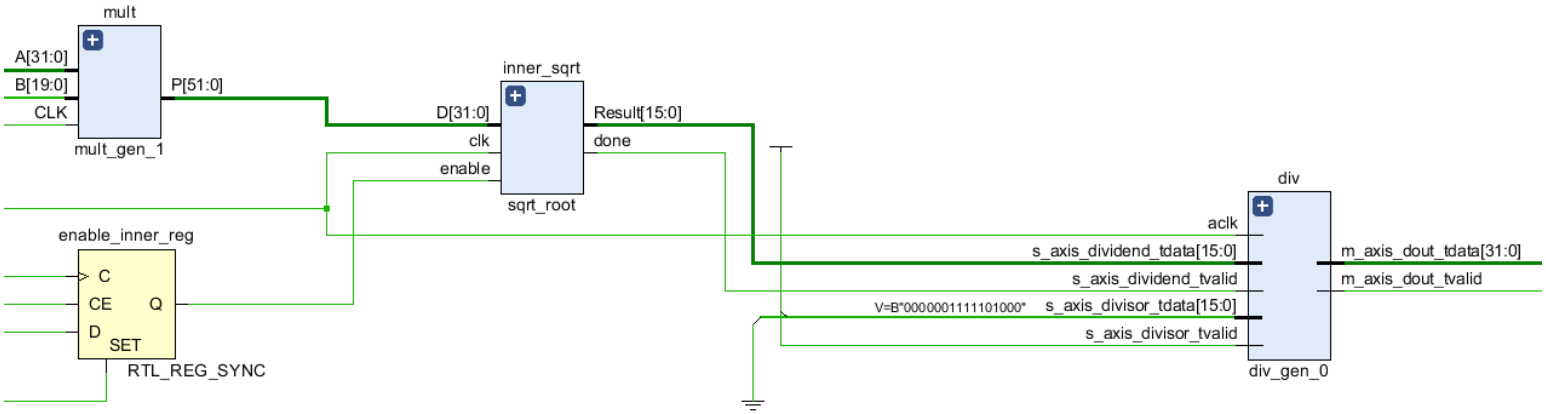
*Total time = 2 * number of iterations * clock cycle = (bit number of D) * clock cycle*

It should be noted that this algorithm works only for decimal numbers. That's why problems occur when fractional numbers are involved. To eliminate this problem, the input value is multiplied by 10^6. At the end of the process, it is divided by 10^3 ($\sqrt{\mathbf{10^6}}$). As can be seen in Figure 3.2 d), while a multiplication module is added to the beginning of the square root module, a division module is added to its end. In addition, a custom made delay unit is added to prevent timing problems in later stages.

When the simulation image in Figure 3.2 e) is examined, it is seen that the fractional D number (in the first line of simulation) is multiplied by 10^6 and transformed into the variable D in the 6^{th} line. Then the square root of this variable is assigned to the Result variable in the 7^{th} line. This value is divided by 10^3 to obtain the Result variable in the 3^{rd} line. The variable D_n is the input D given in pairs, while CR gives the result of each iteration. It can also be seen that the variable k decreases every 2 clock cycles and starts a new iteration. The 32-bit number resulting from the division module is reduced to 16 bits in order to be compatible with the later stages of the study.



(c)



(d)

Figure 3.2 : c) Inner square root module d) Top module with multiplier and divider.

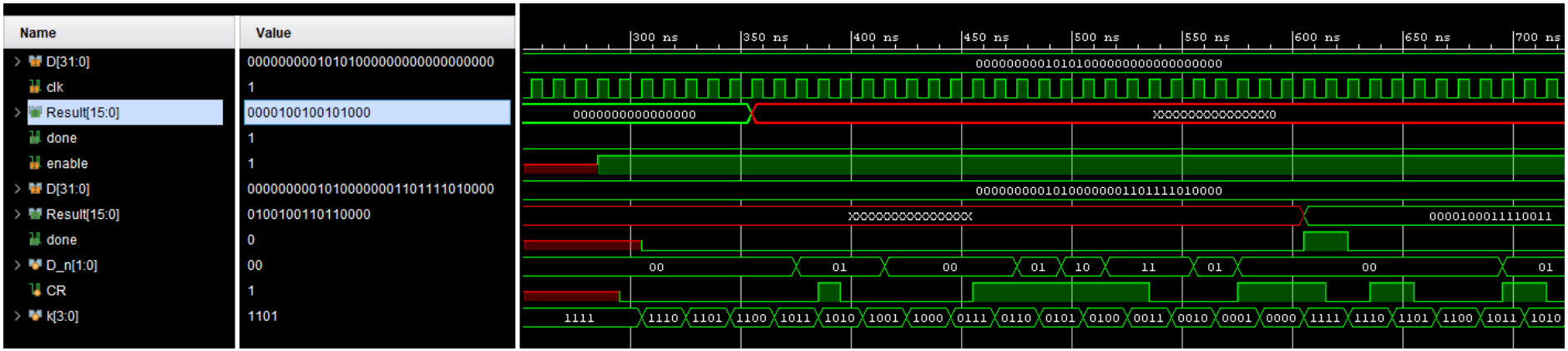


Figure 3.2 : e) Simulation screen on Vivado.

3.3 Triangular Systolic Array (TSA) for Modified Gram Schmidt Theorem

3.3.1 Design

MGS orthogonalization is implemented in the QRD block which is designed by using the Triangular SA (TSA) structure. TSA architecture is created by combining Internal Cells (IC) and Boundary Cells (BC) to form a triangular structure which is depicted in Figure 3.3.1. In this structure, n diagonal (BC) modules, (n (n-1)) / 2 off-diagonal (IC) modules are designed. In each iteration, the Q column vector and the R diagonal element are the output signal from the BC module. R upper diagonal elements are the output signal from the IC module.

The MGS algorithm has two nested for loops depicted in Figure 1.2.2.2. The algorithm is divided into two parts: operations within the main loop and the nested loop. The operations in the main loop are repeated for the number of columns. In other words, iterations depend only on the number of columns. For this reason, it should be repeated once in each line in the created TSA model (Figure 3.3.1) and should be placed in the BC (diagonal cell). Looking at the nested for loop, each time the column value changes, the row value increases up to column number - 1, so when the matrix is considered, there should be (n-1) in the first row. This process needs to be repeated (n (n-1)) / 2 times in total. In this case, the steps that need to be handled multiple times in each line are called an IC (off-diagonal cell). When the architecture is followed downwards, the number of diagonals remains constant whereas the number of off-diagonals decreases. Thus it forms a complete TSA structure.

If it is explained by considering the 4x4 input matrix, boundary cell operations are repeated 4 times for j = 1 : 4 loop. When looking at the loop i = 1: j-1, it is iterated either 1, 2 or 3 times according to the value of j. As a result, there are one diagonal and 3 off-diagonal cells in the first line. When the architecture is followed downwards, the number of diagonals remains constant whereas the number of off-diagonals decreases. Thus it forms a complete TSA structure.

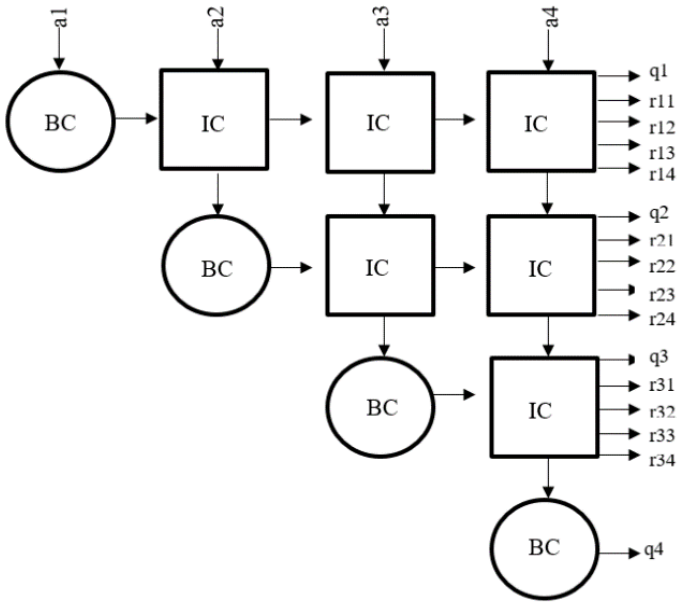
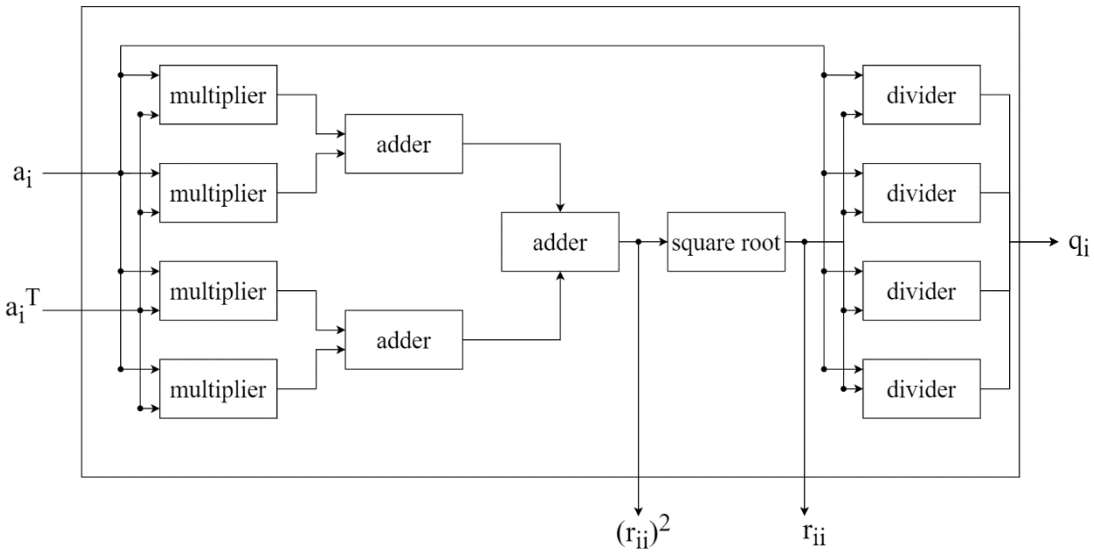
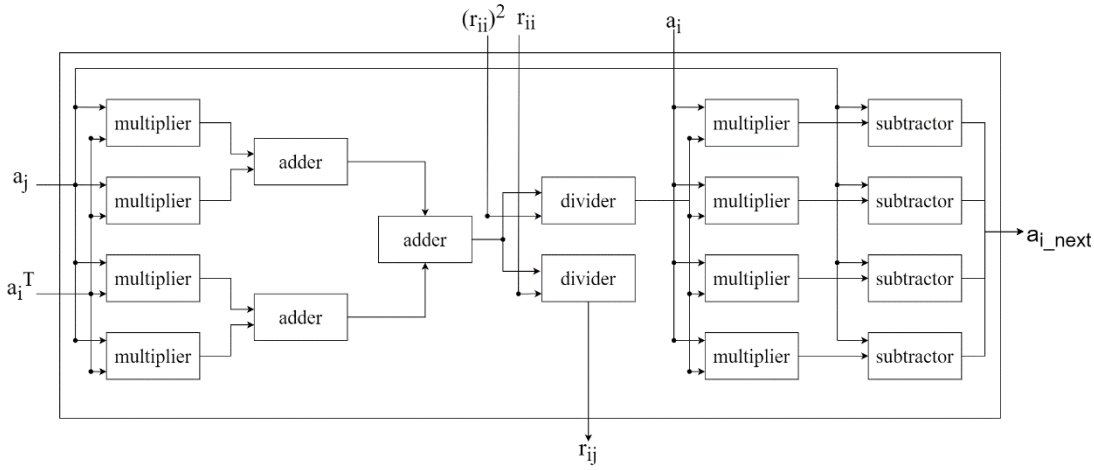


Figure 3.3.1: a) Triangular Systolic Array for Modified Gram Schmidt orthogonalization.

BC (Figure 3.3.1.b) includes multiplication, addition, division, and square root operations. The input ai is the ith column of the input matrix. The output rii (1.2.2.2. b) forms a diagonal element of the R matrix, and qi (1.2.2.2. c) forms a column of the Q matrix. IC (Figure 3.3.1.c) includes all basic arithmetic operations. anew (a_{ji+1}), which is the output of IC, forms the input matrix of the new iteration [4]. Arithmetic IPs in Vivado are used for arithmetic operations in cells. The sign prevents errors by adding two's complement to the beginning of the multiplication and division modules.



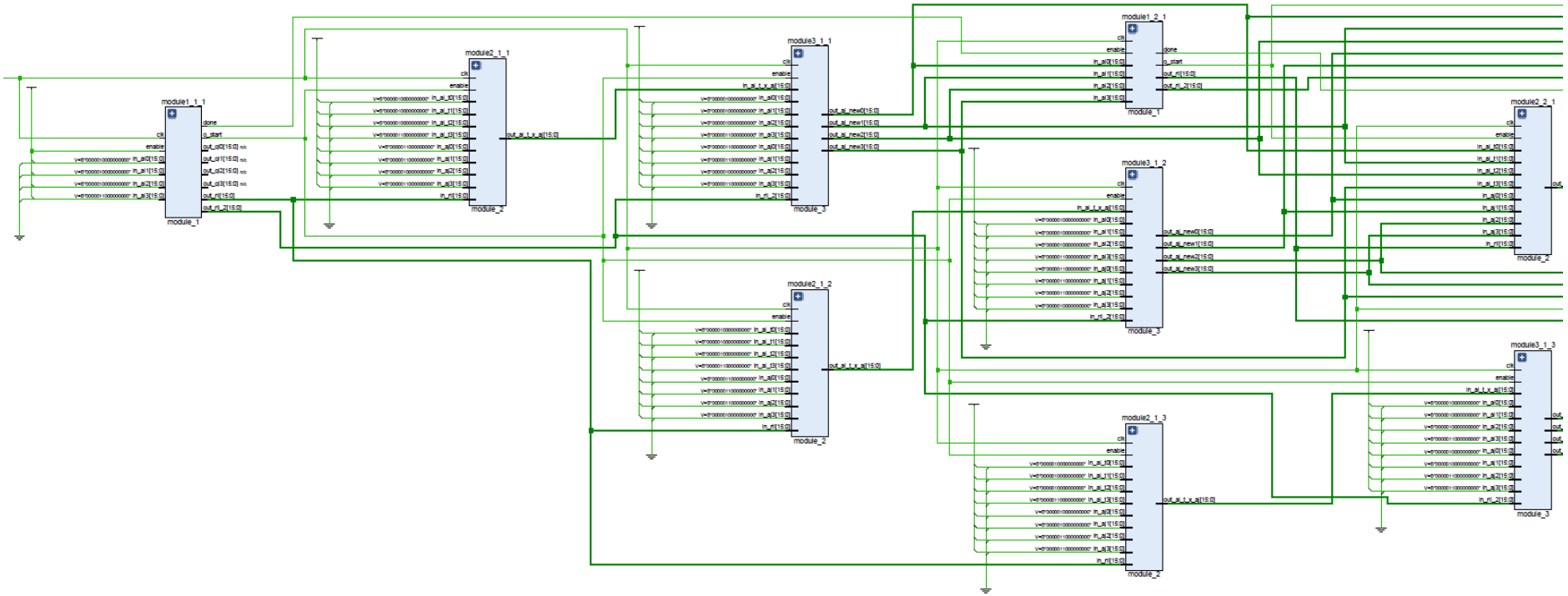
(b)



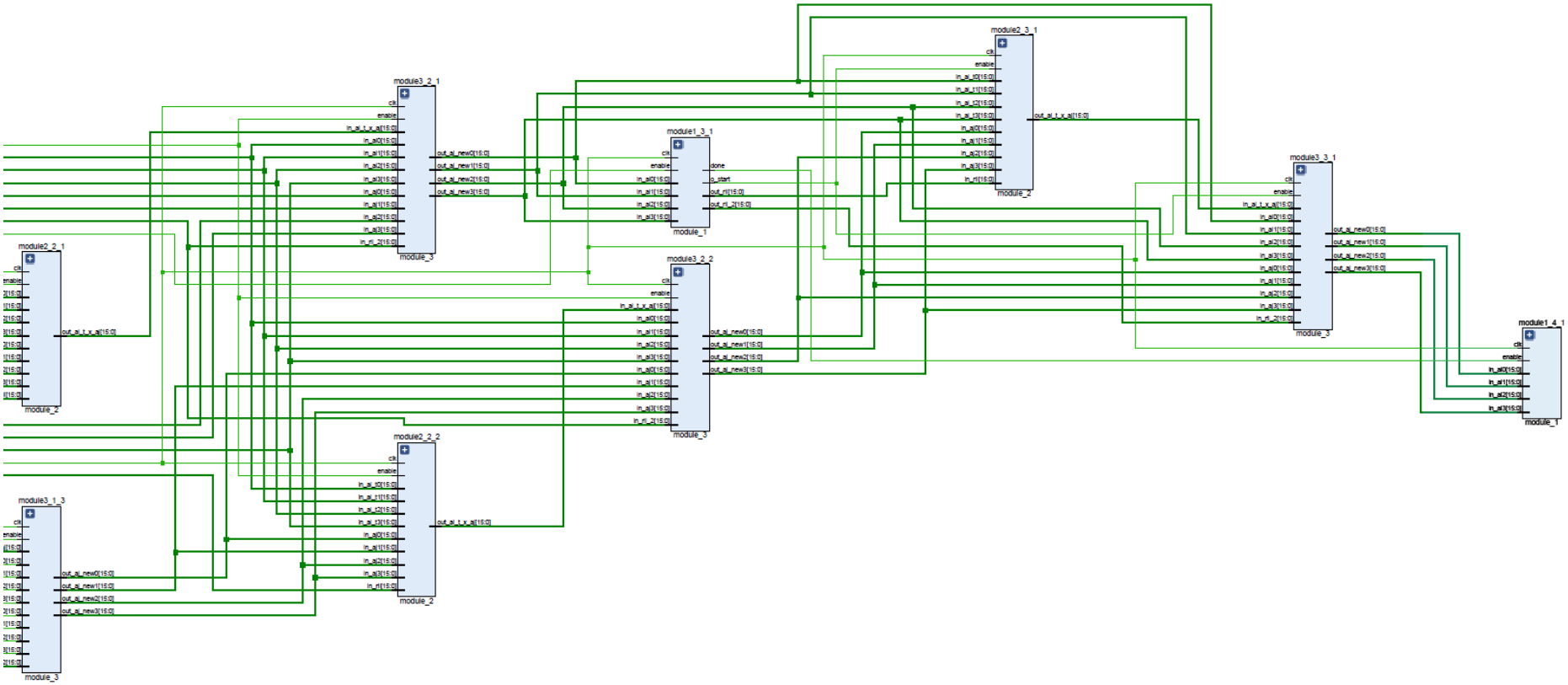
(c)

Figure 3.3.1: b) Boundary Cell (BC) [4] c) Internal Cell (IC) [4].

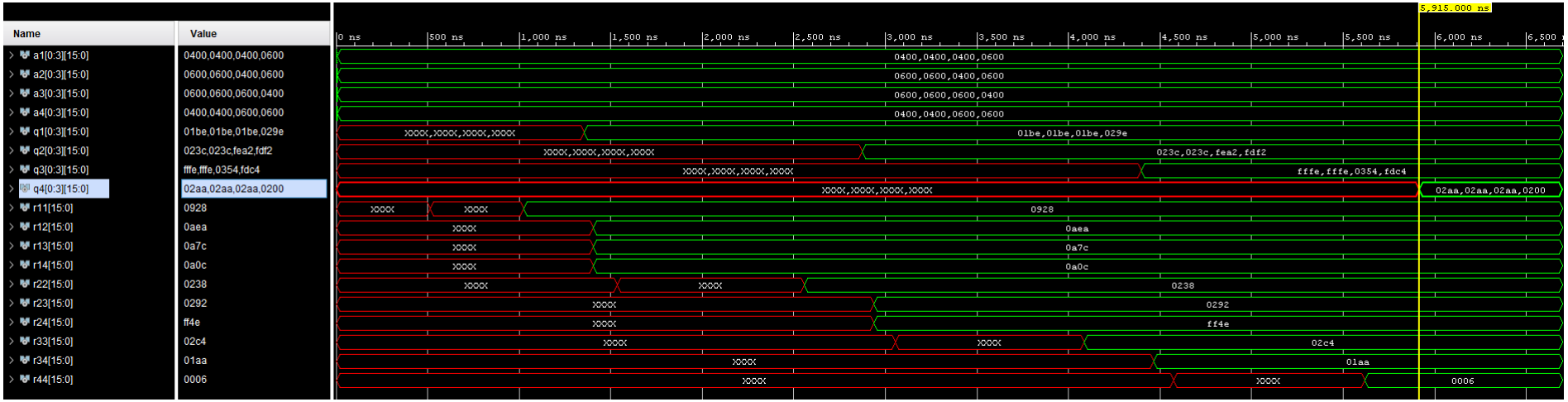
3.3.2 Schematic and simulations on Xilinx Vivado



(a)


module2_3_1
module3_2_1
module1_3_1
module3_3_1
module2_2_1
module3_2_2
module1_4_1
module2_2_2
module3_1_3
module_1
module_2
module_3

(b)



(c)

Figure 3.3.1.2 : Schematic and simulation of Modified Gram Schmidt algorithm a) Left part b) Right part c) Simulation

3.4 Triangular Systolic Array (TSA) for Givens Rotation Theorem

3.4.1 Design

In the QRD block design, the GR algorithm is implemented using the Triangular SA (TSA) structure. Similar to the TSA architecture in the MGS module, a triangular structure shown in Figure 3.3.1.a is created by combining IC and BC. However, the BC and IC modules in this architecture are different for each column. Area usage is reduced by minimizing the circuit in each column. In the BC module, the sin and cos values, which are the elements of the givens rotations matrix, are produced, so it is also called the Givens Generation (GG). In IC modules, the rows of the matrix are updated and called Row Update (RU). In each iteration, the diagonal element R is the output signal from module BC. R upper diagonal elements are the output signal from the IC module. In this structure, n - 1 diagonal (GG) modules, (n (n-1)) / 2 off-diagonal (RU) modules are used.

The GR algorithm given in Figure 1.2.2.1 includes a nested for loop, similar to MGS. But all calculations are done under the inner for loop. That indicates all operations must be done for each lower diagonal element of the input matrix, and thus the architecture cannot be parallelized enough. Therefore, Column-wise Givens Rotation (CGR) is used to adapt the algorithm to the TSA structure. The sin and cos values of the Givens rotation matrix are calculated for each column instead of each lower diagonal element. Thus, the BC module is created. These sin, cos, k, and L values are then given to the RU modules in the same TSA row to update the R matrix. Updates of the R matrix are performed within the RU module. Then the updates create the new matrix.

From the nxn input matrix, (n)x(n-i) sized intermediate matrix is obtained in i^{th} iteration. The 1^{st} row of this matrix forms the i^{th} row of the R output matrix. The remaining (n-i)x(n-i) dimensional part of the intermediate matrix enters the next iteration as the updated matrix.

For a 4x4 matrix example matrix, p3 (norm of columns) from GG1 block output and updates obtained from the RU1 block is placed as the first row. At the same time, the elements below the first element p3 in Column1 are zeroed. Calculated Row1 and Column1 form the first column and row of the output matrix R. Remaining elements of the updated matrix are obtained by updating the input matrix columns using the sin and cos values with the operations shown in Figure 3.4.1.a. These operations are performed by taking the first column and the other columns two by two in the RU1 block. At the same time, the 3x3 part of the updated matrix, excluding the first row and first column, creates the new matrix. With this new matrix, the same operations are repeated similarly to obtain a 2x2 matrix. The new values obtained at this stage form the final elements of the R matrix.

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

$$GX = \begin{bmatrix} p_3 & \frac{x_{11}x_{12}+s_{11}}{p_3} & \frac{x_{11}x_{13}+s_{12}}{p_3} & \frac{x_{11}x_{14}+s_{13}}{p_3} \\ 0 & \frac{x_{11}s_{11}}{p_3p_2} - \frac{x_{12}p_2}{p_3} & \frac{x_{11}s_{12}}{p_3p_2} - \frac{x_{13}p_2}{p_3} & \frac{x_{11}s_{13}}{p_3p_2} - \frac{x_{14}p_2}{p_3} \\ 0 & \frac{x_{21}s_{21}}{p_2p_1} - \frac{x_{22}p_1}{p_2} & \frac{x_{21}s_{22}}{p_2p_1} - \frac{x_{23}p_1}{p_2} & \frac{x_{21}s_{23}}{p_2p_1} - \frac{x_{24}p_1}{p_2} \\ 0 & \frac{x_{31}x_{42}}{p_1} - \frac{x_{41}x_{32}}{p_1} & \frac{x_{31}x_{43}}{p_1} - \frac{x_{41}x_{33}}{p_1} & \frac{x_{31}x_{44}}{p_1} - \frac{x_{41}x_{34}}{p_1} \end{bmatrix}$$

$$= \begin{bmatrix} p_3 & \frac{x_{11}x_{12}+s_{11}}{p_3} & \frac{x_{11}x_{13}+s_{12}}{p_3} & \frac{x_{11}x_{14}+s_{13}}{p_3} \\ 0 & k_1s_{11} - x_{12}l_1 & k_1s_{12} - x_{13}l_1 & k_1s_{13} - x_{14}l_1 \\ 0 & k_2s_{21} - x_{22}l_2 & k_2s_{22} - x_{23}l_2 & k_2s_{23} - x_{24}l_2 \\ 0 & cx_{42} - x_{32}s & cx_{43} - x_{33}s & cx_{44} - x_{34}s \end{bmatrix}$$

Figure 3.4.1: a) Updated matris of 4x4 input matrix after one iteration [12].

GG (Figure 3.4.1 c, d, e) includes multiplication, addition, division, and square root operations. The input x is the ith row and jth column element of the input matrix. The output p3 in GG1 forms the first diagonal element of the R matrix. The output p2 in GG2 and p1 in GG3 forms the second and third diagonal elements of the R matrix respectively. The fourth diagonal element is the row4 update output of the RU3 module. L, k outputs, and sin and cos outputs of Givens Rotation matrix are produced to be used in RU modules. RU (Figure 3.4.1 f, g, h) includes all basic arithmetic operations. Row updates, which are the outputs of RU, forms the input matrix of the new iteration. As with MGS, Arithmetic IPs in Vivado are used for arithmetic operations in cells then the sign errors are prevented by adding two's complements to the beginning of the multiplication and division modules.

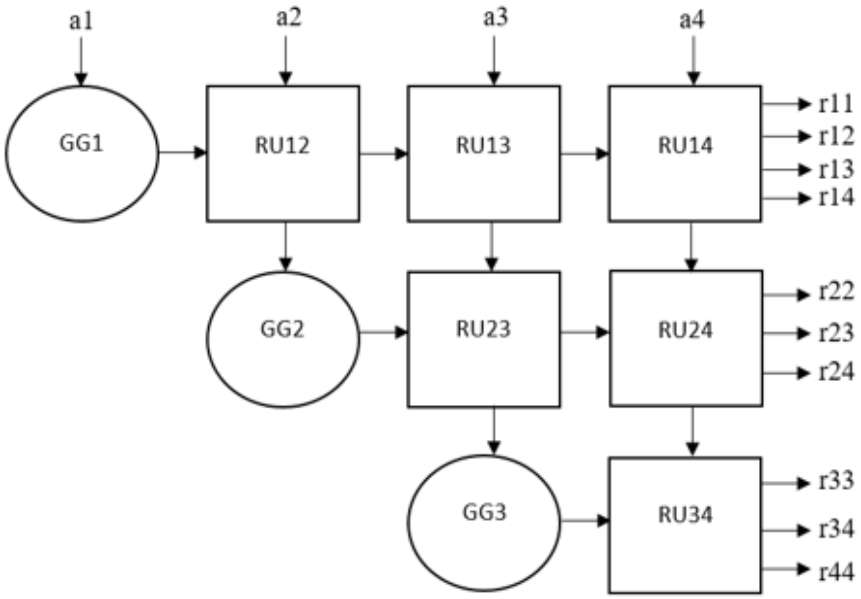
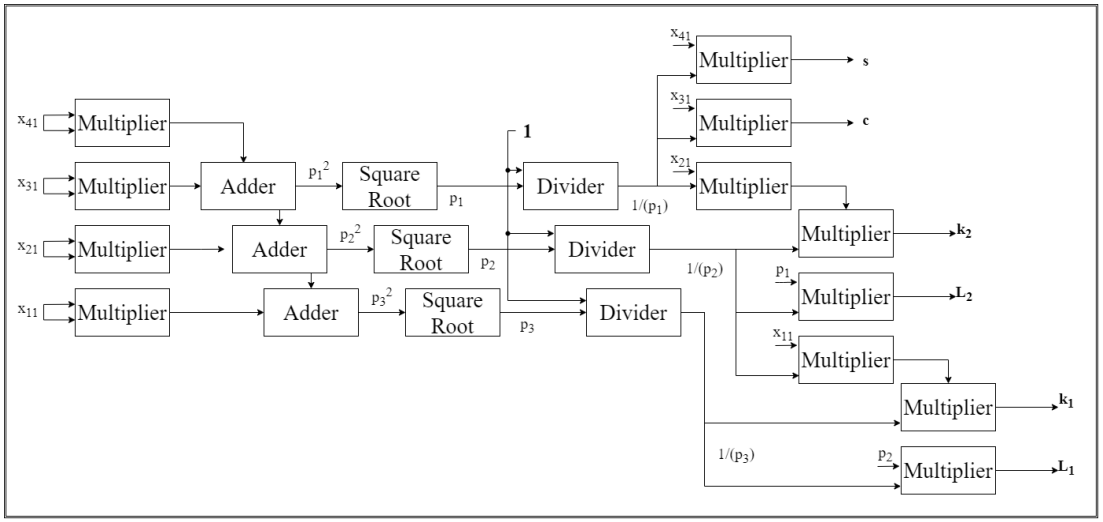
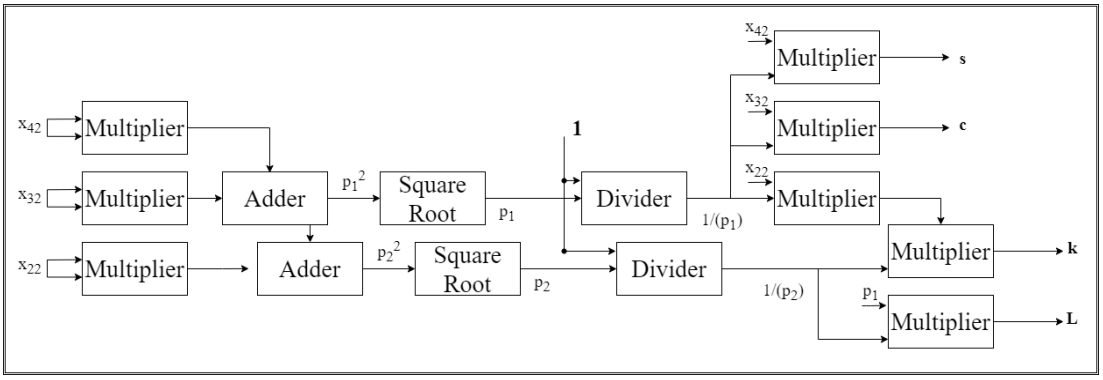


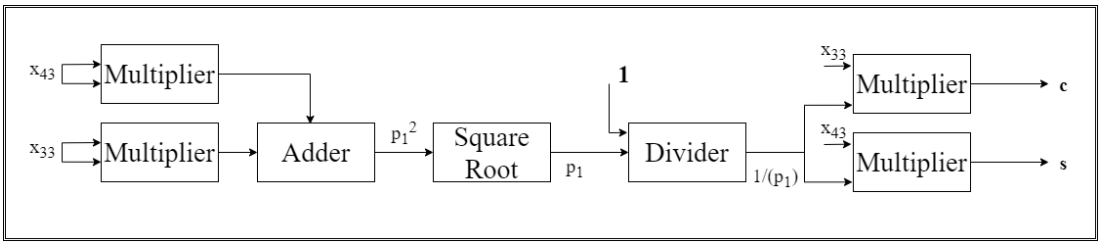
Figure 3.4.1: b) Triangular Systolic Array for Givens Rotations theorem [13].



(c)

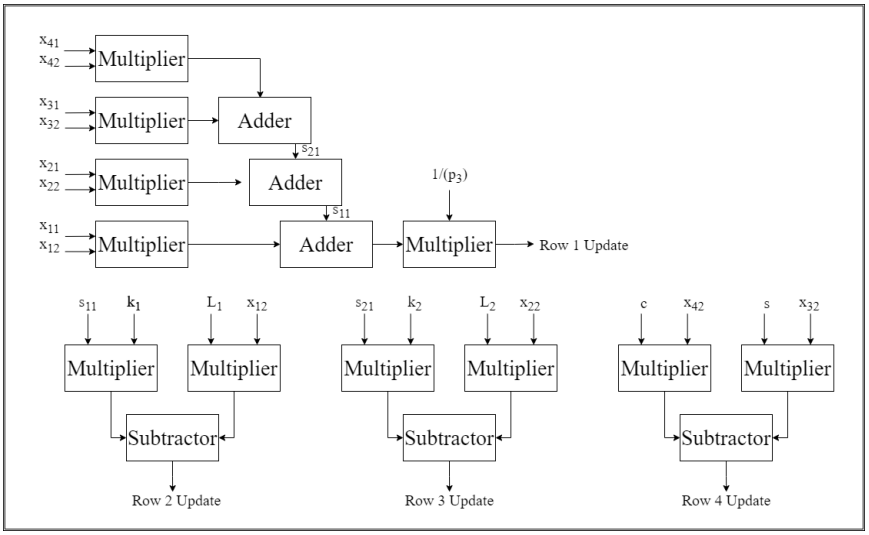


(d)

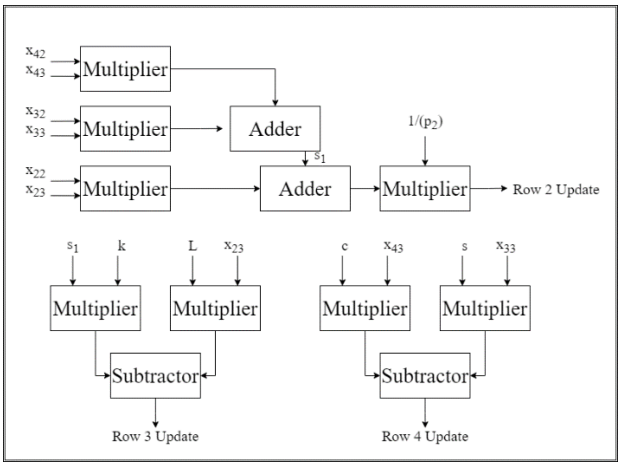


(e)

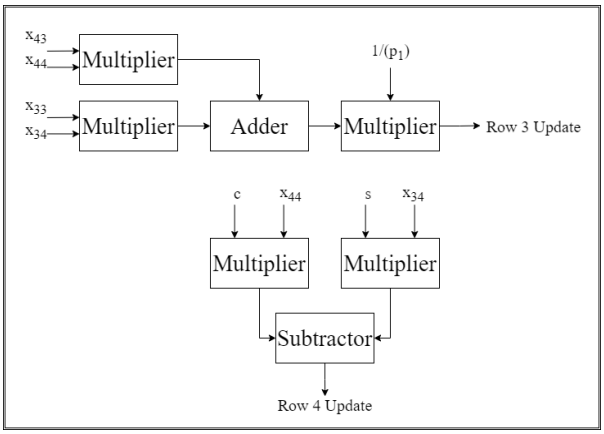
Figure 3.4.1: c) GG1 module [13]. **d)** GG2 module [13]**. e)** GG3 module [13].



(f)



(g)



(h)

Figure 3.4.1: f) RU1 module [13]. **g)** RU2 module [13]. **h)** RU3 module [13].

With the TSA structure, only the R matrix is calculated. Then, the Q matrix is obtained by using the A=QR equation for eigenvalue calculation.

The 1st column is obtained as

$$q_{i1} = \frac{a_{i1}}{r_{11}} \,. \qquad (3.4.1.a)$$

The 2nd column is obtained as

$$q_{i2} = \frac{a_{i2} - q_{i1}r_{12}}{r_{22}} \,. \qquad (3.4.1.b)$$

The 3rd column is obtained as

$$q_{i3} = \frac{a_{i3} - q_{i1}r_{13} - q_{i2}r_{23}}{r_{33}} \,. \qquad (3.4.1.c)$$

The 4th column is obtained as

$$q_{i4} = \frac{a_{i4} - q_{i1}r_{14} - q_{i2}r_{24} - q_{i3}r_{34}}{r_{44}} \,. \qquad (3.4.1.d)$$

While the same operations are performed in parallel for each row, the elements in the same row are calculated sequentially. Thus, the square SA structure is formed.

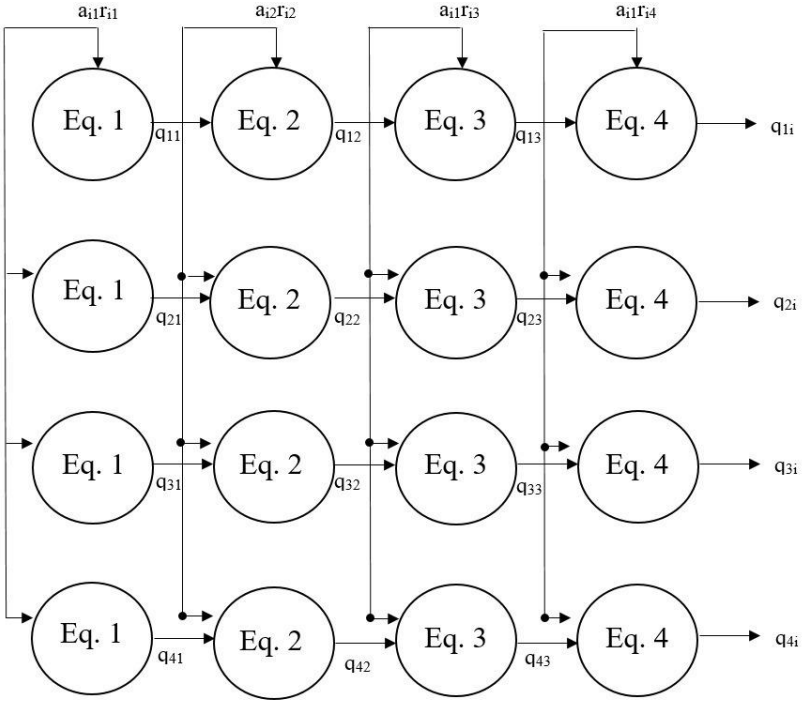
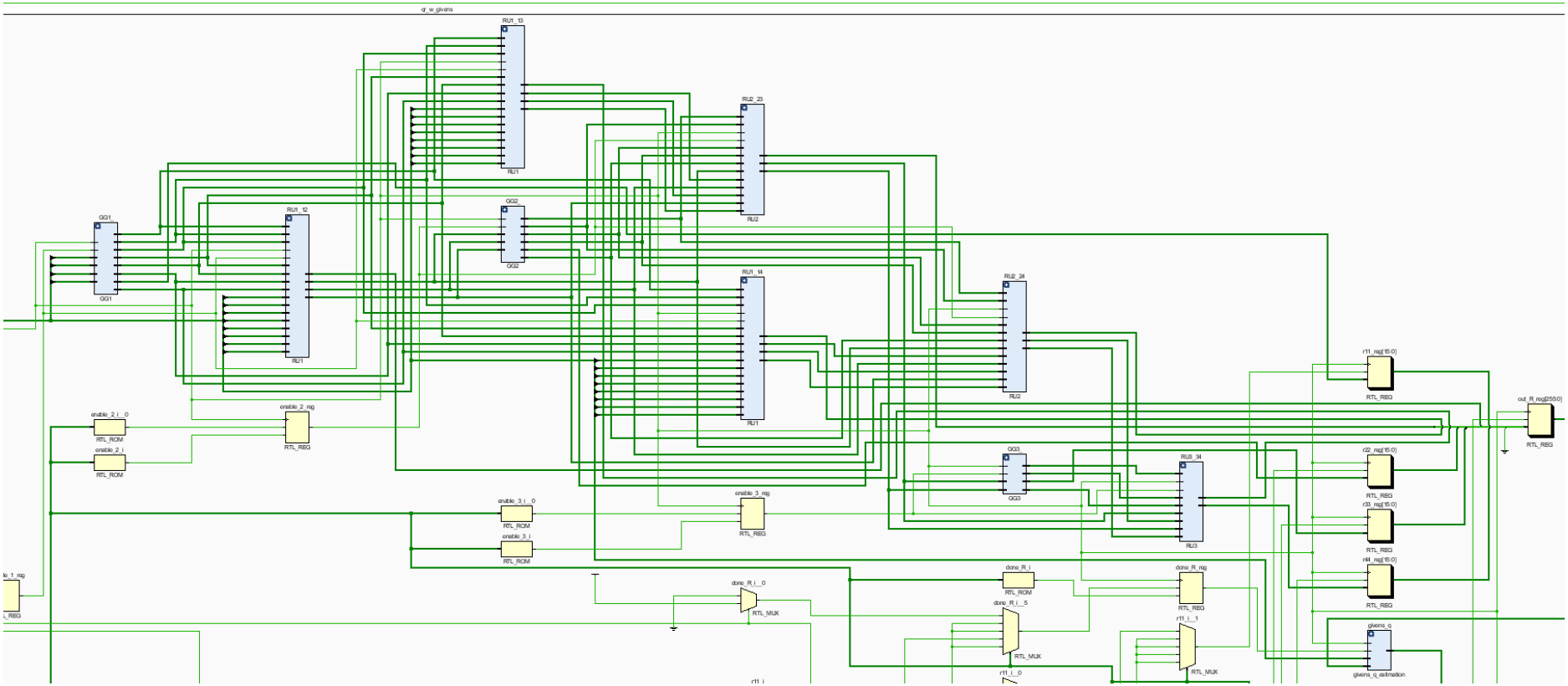
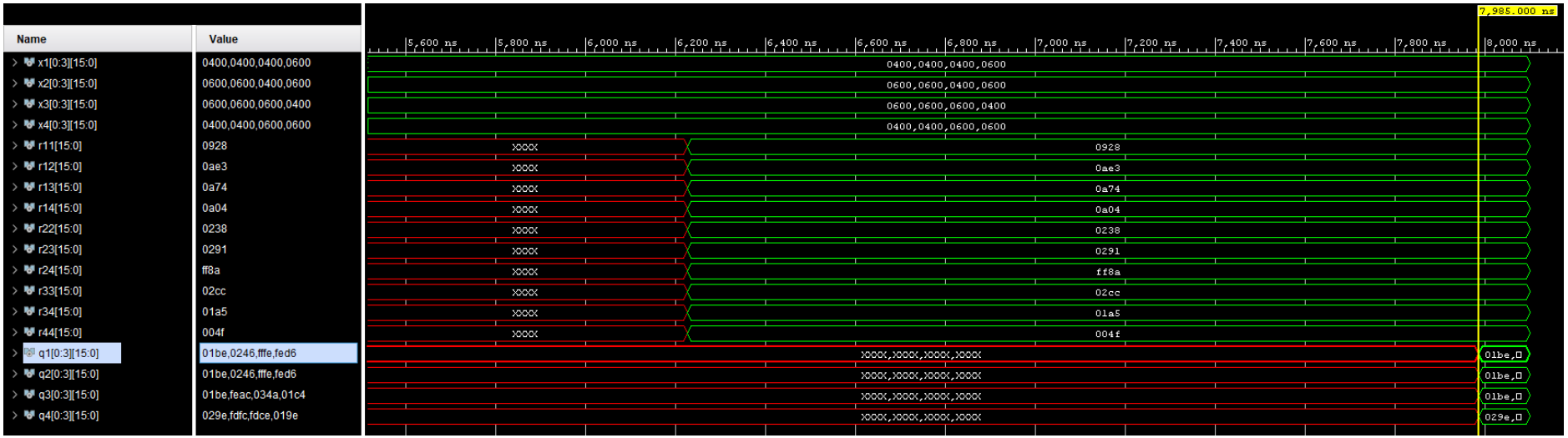


Figure 3.4.1: i) SA structure for Q estimation block.

3.4.2 Schematic and simulations on Xilinx Vivado



(a)



(b)

Figure 3.4.2: Schematic and simulation of Modified Gram Schmidt algorithm a) Schematic b) Simulation

3.5 Control Diagram of Eigenvalue Calculation

The QR algorithm consists of QRD and Matrix Multiplication blocks. The input matrix enters the QRD block so that the Q and R matrices are calculated. Then these matrices enter the Matrix Multiplication block and perform the R * Q operation. The resulting matrix is given to the Diagonal Check block. In this block, lower-diagonal elements (except for $((i+1),i)^{th}$ elements) are checked. If any non-zero element is encountered, the matrix is sent back to the QRD block for a new iteration. The iteration number increases with the matrix size, and it is higher in asymmetric matrices than in symmetric ones. When the iterations are completed, the diagonals show the eigenvalues. Two consecutive blocks are added for the check of non-zeroed elements and the calculation of complex conjugate pairs. In the Check Complex Conjugate Pairs block, $((i+1),i)^{th}$ elements are controlled. If any non-zero element is encountered, it indicates a complex conjugate pair. The 2x2 matrix part containing this element enters the Eigenvalue Calculation for the 2x2 block. Thus the eigenvalue calculation is completed. The control diagram of this proposed system is depicted in Figure 3.5. In this study, two different QRD blocks that calculate QR were implemented. Then, the eigenvalue estimation was handled according to the control diagram. Thus, two separate eigenvalue calculation blocks with only different QRD blocks were designed and compared.

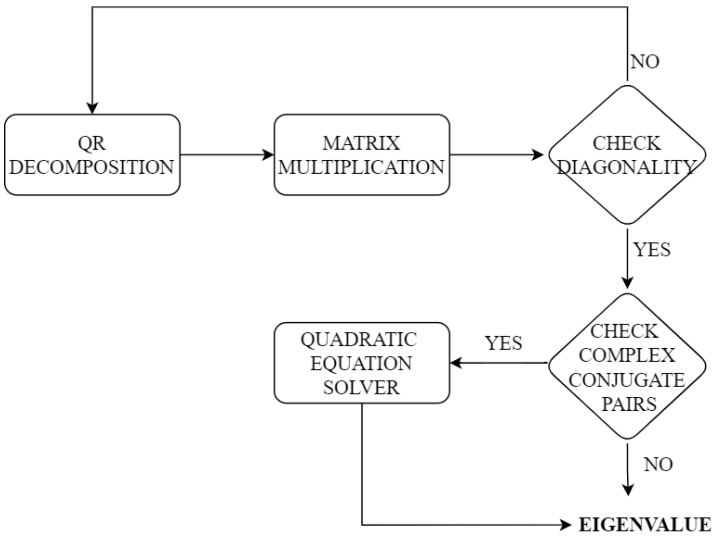


Figure 3.5: Control diagram of eigenvalue calculation in the proposed system.

3.6 Matrix Multiplication Architecture

In the Matrix Multiplication block, the calculation of the ijth element of the result is obtained by multiplying the i^{th} row of R and the j^{th} column of Q. In this block, operations are handled by 4x4 matrices. For this reason, four multiplication and three addition operations are performed for each element of the result as an inner module. This module is implemented in parallel for 16 matrix elements to form the overall Matrix Multiplication block. Due to parallelization, time efficiency is provided, but area utilization is compromised.

3.7 Diagonality and Complex Conjugate Pairs Check Blocks

The Diagonality Check block checks if the (4,1), (3,1), and (4,2) elements of the 4x4 A matrix equal to zero. The iteration given in the control diagram (Figure 1) is continued until the matrix takes the upper triangular form. The diagonal elements of the resulting matrix give the eigenvalues. Then the matrix is given to the Complex Conjugate Pairs Check block. If $((i + 1),i)^{th}$ elements are non-zero, eigenvalues appear as complex conjugate pairs. QR Algorithm fails to equate these elements to zero, so calculations need to be handled in another block. For these calculations, a 2x2 matrix is taken where the non-zeroed $(i+1,i)^{th}$ element of the input matrix is the $(2,1)^{th}$ element of the new matrix. The diagonal elements of the 2x2 matrix are $(i,i)^{th}$ and $(i +1, i+1)^{th}$ elements of the input matrix, respectively. The $(1,2)^{th}$ element is the $(i,i+1)^{th}$ element.

3.8 Eigenvalue Estimation for Complex Conjuge Pairs

To determine the complex conjugate eigenvalue pairs, the equations given in Figure 2.2.2 are implemented in digital design. When Figure 3.8 is examined, square root, multiplication, addition, subtraction, shift operation, and 2's complementary operation modules are used to solve the equation. The square root module is explained in detail in section 3.2. To perform multiplication, addition, and subtraction, Vivado IP modules are added to the design and then customized to accommodate negative values. In the 2's complementary module ((16'hffff xor input_value) + 1) operation is implemented. The obtained eigenvalues are in (eig_reel + eig_complex) and (eig_reel - eig_complex) form.

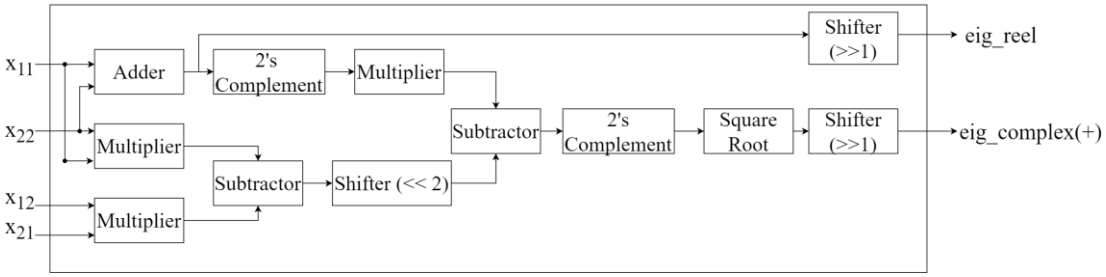


Figure 3.8: Quadratic equation solver.

3.9 Results and Comparison

Asymmetric eigenvalue calculation based on QR algorithm was performed separately using two different QRD blocks. Behavioral simulation and post-synthesis reports are examined and compared for two systems in this section.

In Table 3.9.a, the operations used for the n by n input matrix are given and compared. The number of operations is given depending on the variable n. Looking at the differences, MGS uses fewer operations from 8 onwards, and more multiplication and division modules are used when n is less than 7. For example, for a 4x4 matrix, the MGS module uses 64 multiplication, 24 addition, 24 subtraction, 28 division, and 4 square root modules. GR uses 48 multiplications, 12 additions, 30 subtractions, 22 divisions, and 6 square roots. Thus, it is seen that operations other than square root and subtraction are used more in MGS. The main reason for this is that the internal structures of the BC and IC modules of the GR algorithm are getting smaller and smaller.

In Table 3.9.b, the area usage of the modules made for the 4x4 matrix is given and compared. More LUTs were used when implementing GR, whereas MGS used more registers.

$$Input\ matrix: \begin{bmatrix} 1 & 1.5 & 1.5 & 1 \\ 1 & 1.5 & 1.5 & 1 \\ 1 & 1 & 1.5 & 1.5 \\ 1.5 & 1.5 & 1 & 1.5 \end{bmatrix} \tag{3.9}$$

The eigenvalues of the input matrix given in Eq. 10 are shown in Table 3.9.c. These values are compared with MATLAB outputs to determine the error amounts. For this

specific example input matrix, the GR eigenvalue outputs are calculated closer to MATLAB results when compared with MGS error results.

The general schematic of the eigenvalue calculation system is given in Figure 3.9.a. The schematic includes the black boxes of the QRD, Matrix Multiplication, and Quadratic Equation Solver modules, as well as the registers and combinational circuits of the finite state machine structure used to create the control structure.

The simulations of the eigenvalue calculation handled by GR and Givens Rotation are given in Figures 3.9.b and 3.9.c, respectively. The signals of the A(MGS) or X(GR) input and Q columns, and R matrices elements are shown, respectively. As seen in the figure, it takes four iterations for the A_new matrix to become upper diagonal after the exit_flag is set. The diagonals of the A_new matrix give real eigenvalues. Then, the complex conjugated pair is controlled. eig_comp and eig_real are calculated by taking a 2x2 matrix containing this pair.

When the simulation is analyzed, there are fewer iterations in GR simulation and therefore the results come in a shorter time. However, this does not imply that GR is faster since a GR iteration takes about 80 clock cycles, while an MGS takes about 60 clock cycles. However, since an equal amount of time is determined for each iteration in the top module FSM, GR, having fewer iteration counts, achieved results faster. The simulations use a 100 MHz clock frequency. That's why in the MGS simulation, it takes 330 clock cycles to get the result. However, if the iteration number is multiplied by the time required for an iteration, 240 clock cycles are found. When combined with these other modules, 250 clock cycles can be found, which is close to the output time value of GR.

Table 3.9.a : Number of operations and comparision for n by n input matrix.

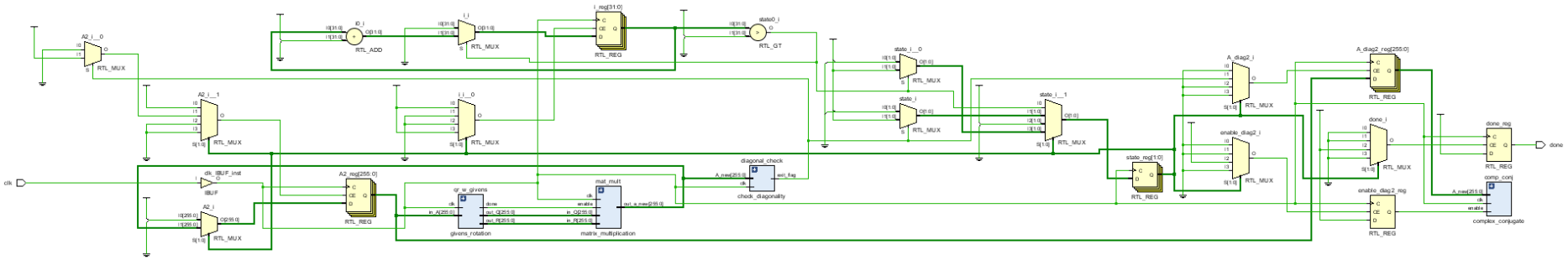
Operation	Givens Rotation	Modified Gram Schmidt	Matrix Multiplication Block	Quadratic Equation Solver	Total for GR	Total for MGS	Differences (GR - MGS)
Multiplication	$9n^2/2-7n/2-2$	$4n^2$	$4n^2$	3	$17n^2/2 - 7n/2 +1$	$8n^2+3$	$n^2/2 - 7n/2 -2$
Addition	n^2-n	n^2+2n	$3n^2$	1	$4n^2-n+1$	$4n^2+2n+1$	$-3n$
Subtraction	$5(n^2-n) /2$	$2n^2-2n$	-	2	$5(n^2-n) /2 + 2$	$2n^2-2n +2$	$(n^2-n)/2$
Division	$3n^2/2 - n/2$	n^2+3n	-	-	$3n^2/2 - n/2$	$3n^2/2 - n/2$	$n^2/2-7n/2$
Square Root	$(n^2-n)/2$	n	-	1	$(n^2-n)/2 +1$	n + 1	$n^2/2-3n/2$
Shifter	-	-	-	3	3	3	-
2’s Complement	-	-	-	2	2	2	-

Table 3.9.b : Area usage and comparision for 4 by 4 input matrix..

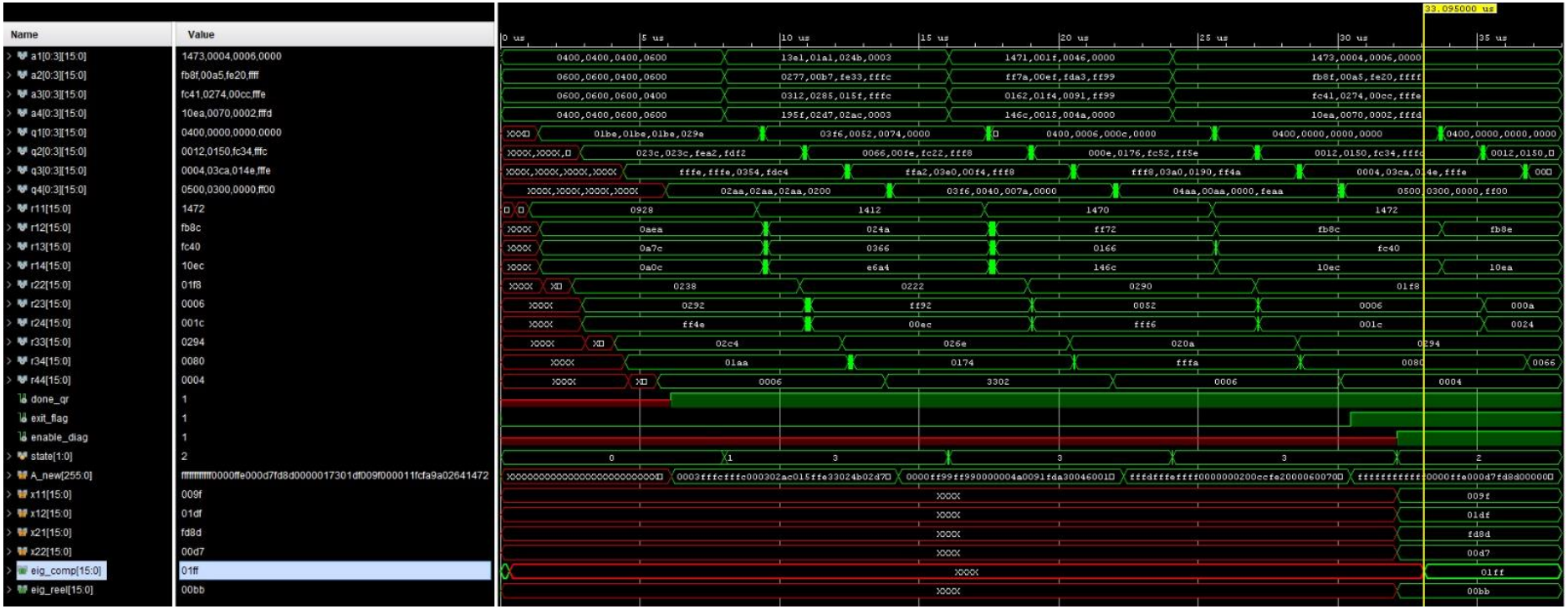
Area	Givens Rotation	Modified Gram Schmidt	Matrix Multiplication Block	Quadratic Equation Solver	Total for GR	Total for MGS	Differences (GR - MGS)
Slice LUT	87265	58996	26301	8151	122976	93728	29248
Slice Register	71082	73255	9408	7419	88599	90430	-1831

Table 3.9.c : Eigenvalue comparision with MATLAB for 4 by 4 matrix.

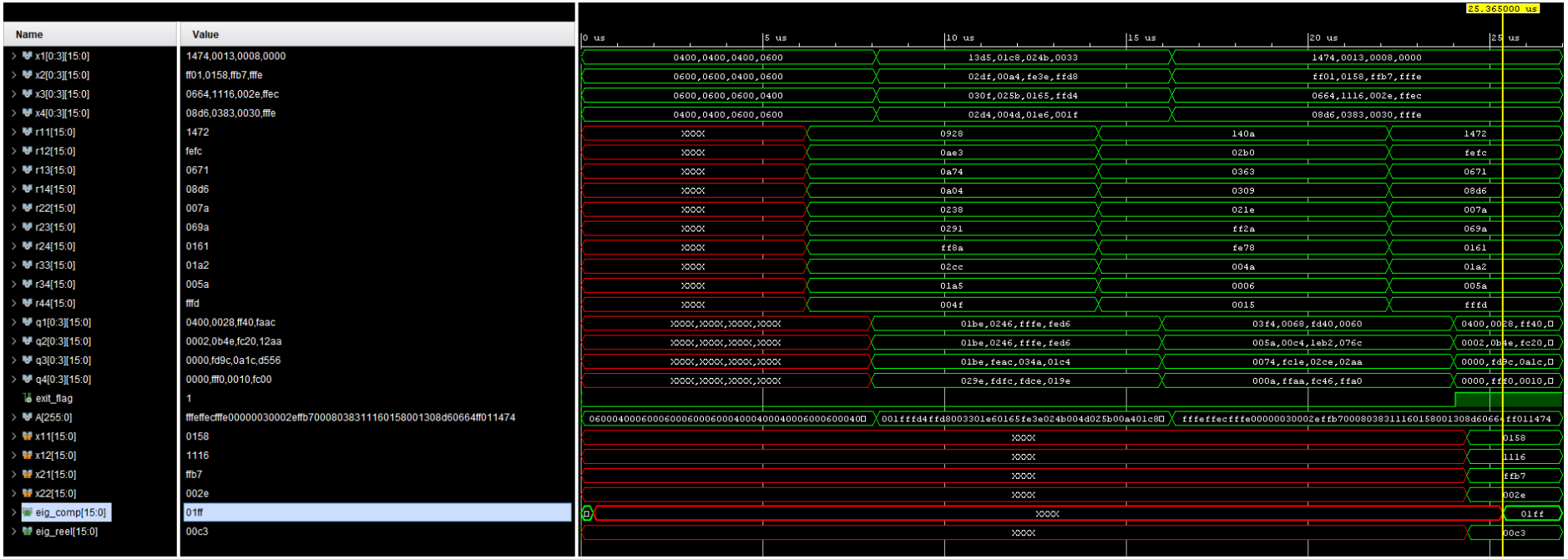
Eigenvalues	MATLAB Implemented QR Algorithm with GR	MATLAB Implemented QR Algorithm with MGS	Digital Implemented QR Algorithm with GR	Digital Implemented QR Algorithm with MGS	Error for GR	Error for MGS
Eigenvalue 1	5.1227 + 0.0000i	5.1227 + 0.0000i	5.1132 + 0.0000i	5.1113 + 0.0000i	0.0095 + 0.0000i	0.0114+ 0.0000i
Eigenvalue 2	0.1887 + 0.5307i	0.1887 + 0.5307i	0.1904 + 0.4990i	0.1826 + 0.4990i	-0.0017 + 0.0317i	0.0060+ 0.0317i
Eigenvalue 3	0.1887 - 0.5307i	0.1887 - 0.5307i	0.1904 - 0.4990i	0.1826 - 0.4990i	-0.0017 - 0.0317i	0.0060 - 0.0317i
Eigenvalue 4	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0009 + 0.0000i	0.0000 + 0.0000i	-0.0009 + 0.0000i	0.0000+ 0.0000i


A2_i__0
RTL_MUX
i0_i
RTL_ADD
i_i
RTL_MUX
i_reg[31:0]
RTL_REG
state0_i
RTL_GT
A2_i__1
i_i__0
state_i__0
state_i
state_i__1
A_diag2_i
A_diag2_reg[255:0]
clk
clk_IBUF_inst
IBUF
A2_i
A2_reg[255:0]
gr_w_givens
givens_rotation
mat_mult
matrix_multiplication
diagonal_check
check_diagonality
state_reg[1:0]
enable_diag2_i
done_i
done_reg
done
enable_diag2_reg
comp_conj
complex_conjugate

(a)


Name
Value
a1[0:3][15:0]
1473,0004,0006,0000
a2[0:3][15:0]
fb8f,00a5,fe20,ffff
a3[0:3][15:0]
fc41,0274,00cc,fffe
a4[0:3][15:0]
10ea,0070,0002,fffd
q1[0:3][15:0]
0400,0000,0000,0000
q2[0:3][15:0]
0012,0150,fc34,fffc
q3[0:3][15:0]
0004,03ca,014e,fffe
q4[0:3][15:0]
0500,0300,0000,ff00
r11[15:0]
1472
r12[15:0]
fb8c
r13[15:0]
fc40
r14[15:0]
10ec
r22[15:0]
01f8
r23[15:0]
0006
r24[15:0]
001c
r33[15:0]
0294
r34[15:0]
0080
r44[15:0]
0004
done_qr
1
exit_flag
1
enable_diag
1
state[1:0]
2
A_new[255:0]
ffffffff0000ffe000d7fd8d0000017301df009f000011fcfa9a02641472
x11[15:0]
009f
x12[15:0]
01df
x21[15:0]
fd8d
x22[15:0]
00d7
eig_comp[15:0]
01ff
eig_reel[15:0]
00bb
33.095000 us
0 us
5 us
10 us
15 us
20 us
25 us
30 us
35 us

(b)



(c)

Figure 3.9: Schematic and simulation of eigenvalue calculation **a)** Schematic **b)** Simulation with MGS **c)** Simulation for eigenvalue calculation input $a_{1,2,3,4}$ matrix. $q_{1,2,3,4}$ for columns of orthogonal Q matrix and $r_{i,j}$ upper triangular R matrix elements. Result matrix A_new (upper triangular and diagonals are real eigenvalues). Complex conjugate pair eig_reel and eig_comp.

4. REALISTIC CONSTRAINTS AND CONCLUSIONS

In this paper, the digital system design of eigenvalue calculation for asymmetric matrices using the QR algorithm is studied. At the end of this study, a complete digital architecture of eigenvalue calculation is implemented by combining and customizing the reference works. MGS orthogonalization and Givens Rotation are chosen for the QRD block. The complete architecture is adapted to require only mathematical and square root calculations. While designing this block, TSA architecture is taken as a base to achieve time efficiency by parallelization. However, since optimization is not set as an interest of this work, IP cores are used. The operation speed can be increased further by optimizing the IP cores, block delays, and frequency. Based on the example given in the study, although Givens Rotation has more area usage, the algorithm reached the result faster and with more accurate values. Gram Schmidt algorithm, on the other hand, has an easier architecture to implement in TSA. In addition, the comparison results may vary according to the matrix size. The results obtained from the eigenvalue calculation design using the MGS algorithm have been submitted to the Cellular Nanoscale Networks and Applications (CNNA) 2021 Conference. The Gram Schmidt based implementation has been added to GitHub and can be found at this link: https://github.com/elifozturkk/Eigenvalue-Computing.git

4.1 Practical Application of this Project

As mentioned in the introduction, eigenvalue estimation is fundamental in the communication technologies and digital signal processing areas such as MIMO detection for wireless communication systems, adaptive beamforming, estimating the DOA. Since this study is done for real-valued matrices, it can be easily integrated into systems that calculate eigenvalues. An important plus is that it gives results for both symmetric and asymmetric matrices.

4.2 Realistic Constraints

While doing the thesis study, social, environmental and economic effects should be made in accordance with international standards and health and safety concerns. These topics are explained in sub-headings below.

4.2.1 Social, environmental and economic impact

Eigenvalue calculation is used in many systems, the efficiency of most systems can be increased with a low power consumption, fast and integrated digital circuit targeted in the project.

4.2.2 Cost analysis

In the project 2 engineers worked for 9 months.

Requirements for the project to progress:

- An office where engineers can work comfortably
- 4 workstations and equipment

The cost of the project is as follows:

- 9-month laboratory rent: 9 * 3.000 ₺ = 27.000₺
- 2 Workstations and equipment: 2 * 10.000 ₺ = 20.000₺
- Employee salaries total monthly: 2 * 5000₺ = 10.000₺
- Employee salaries (9 months): 90.000₺
- Total Cost: 137,000₺

4.2.3 Standards

This project is a prototype development work. Employees have gone through a process that is respectful and sensitive to basic engineering ethics.

4.2.4 Health and safety concerns

There is no risk of the product that will emerge at the end of the project. It does not pose health and safety risks during its use.

4.3 Future Work and Recommendations

The primary aim of the study is to design a digital system that calculates eigenvalues for asymmetric matrices. Optimization studies may be a focus on the future in terms of space usage and working speed. Timing efficiency can be achieved by increasing

the frequency speed. The basic operations used by the modules can be custom-designed instead of using IP cores. In addition, the study has been done for real-valued matrices and can be improved for complex-valued matrices.

REFERENCES

[1] Bravo, I. Vázquez, C. Gardel, A. Lázaro, J. L. and Palomar. E, (2015). *"High Level Synthesis FPGA Implementation of the Jacobi Algorithm to Solve the Eigen Problem,"* Mathematical Problems in Engineering, vol. 2015, pp. 1–11.

[2] Armay, A. O. Habili, R. nd Tallushi, S. (2020). *"Hardware Efficient Architecture to Solve the Eigenvalue Problem,"* thesis.

[3] Dubey, S.S. Shrestha, R. and Chowdhury, S. R. (2016). *"A novel architecture for computing eigenvalues of matrix for high speed applications,"* 2016 IEEE Annual India Conference (INDICON)

[4] Alhamed A. and Alshebeili, S. (2016).*"FPGA implementation of complex-valued QR decomposition,"* 2016 5th International Conference on Electronic Devices, Systems and Applications (ICEDSA).

[5] Boonyi, K. Tagapanij, J. and Boonpoonga, A. (2014). *"FPGA-based hardware/software implementation for MIMO wireless communications,"* 2014 International Electrical Engineering Congress (iEECON).

[6] Chen, D. and Sima, M. (2011). *"Fixed-Point CORDIC-Based QR Decomposition by Givens Rotations on FPGA,"* 2011 International Conference on Reconfigurable Computing and FPGAs.

[7] Maltsev, A. Pestretsov, V. Maslennikov, R. and Khoryaev, A. (2006).

"Triangular Systolic array with reduced latency for QR decomposition of complex matrices," 2006 IEEE International Symposium on Circuits and Systems.

[8] Aslan, S. Niu, S. and Saniie, J. (2012).*"FPGA implementation of fast QR decomposition based on givens rotation,"* 2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS).

[9] Ren M. and Ma, X. (2013). *"Cordic-based Givens QR decomposition for MIMO detectors,"* thesis.

[10] Kung, H. T. (1982). *"Why Systolic Architectures,"* to appear in Computer Magazine.

[11] Putra, R. V. W. (2013). *"A Novel Fixed-Point Square Root Algorithm and Its Digital Hardware Design"* 2013 International Conference on ICT for smart society, 1-4.

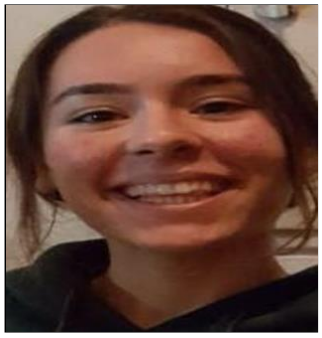
[12] Merchant, F. Chattopadhyay, A. Garga,G. Nandy, S. K. Narayan R. and Gopalan, N. (2014). *"Efficient QR Decomposition Using Low Complexity Column-wise Givens Rotation (CGR),"* 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems, 2014, pp. 258-263, doi: 10.1109/VLSID.2014.51.

[13] Vijayan, V. Jiavana, K.F. (2015). *"Implementation of QR Decomposition for MIMO Detection,"* International Journal of Engineering Research & Technology (IJERT) vol. 4.

CURRICULUM VITAE

Name Surname	**: Elif ÖZTÜRK**
Place and Date of Birth	**: Turkey, 13.09.1999**
E-Mail	**: eozturk17@itu.edu.tr / elifozturk1109@gmail.com**

CURRICULUM VITAE



Name Surname : İlayda KÖSEOĞLU

Place and Date of Birth : Turkey, 21.03.1998

E-Mail : koseoglui18@itu.edu.tr / ilaydakoseoglu98@hotmail.com