Realization of Area Efficient QR Factorization Using Unified Division, Square Root, and Inverse Square Root Hardware

Semih Aslan, Erdal Oruklu, and Jafar Saniie

Electrical and Computer Engineering Department
Illinois Institute of Technology
Chicago, Illinois, 60616, USA

Abstract — **The QR factorization is used in many signal processing and communication applications such as echo cancellation, adaptive beamforming and multiple-input-multiple-output (MIMO) systems. However, division, square root and inverse square root operations required by the QR algorithm are very difficult to implement because they are computationally slow and area-consuming arithmetic operations. This paper presents unified hardware architecture for fast, area efficient QR factorization based on the Householder transformation. Newton-Raphson, and Goldschmidt algorithms are used for fast division, square root and inverse square root blocks. By using a unified architecture, area and power requirements for QR factorization are reduced without decreasing overall speed. The design and implementation of the proposed hardware is presented with synthesis results based on FPGA hardware.**

I. INTRODUCTION

There are many matrix factorizations used in DSP, digital communication and wireless applications such as; the QR factorization [1], LU factorization, Cholesky and Householder transformations and Givens Rotations. These transformations [2] can be used to calculate Eigen values and least-square problems. These transformations require many arithmetic operations such as addition, subtraction, multiplication, division, square root (sqrt), and inverse sqrt.

Division is one of the most difficult arithmetic operations to implement into hardware. The most common division methods are SRT, Goldschmidt, CORDIC and Newton-Raphson algorithms [3]. The "Pentium Bug" problem in particular made it clear that division needs extra attention and time for design and testing.

This paper contains two substantive sections. First, 8-bit unified division, sqrt, and inverse sqrt blocks are designed using Newton-Raphson and Goldschmidt algorithms. These blocks are then compared in terms of area and speed and power. Goldschmidt and Newton-Raphson methods have issues with throughput but have low latency [4]. Finally, this unified division, sqrt and inverse sqrt block is integrated into an area-efficient QR factorization. Furthermore this proposed area-efficient design is optimized for speed by implementing variable frequency block design. For performance evaluation and synthesis, the QR algorithm is implemented with Xilinx Spartan 3E FPGA.

The main focus of this paper is to implement a unified division, sqrt and inverse sqrt block into a QR algorithm by focusing on fixed iteration division, sqrt and inverse sqrt analysis and design. The number of iterations [3][4][5], initial estimation [6] and error analysis [6][7][8] are important to reduce latency and number of iterations.

II. DIVISION, SQRT AND INVERSE SQRT DESIGN

A. NEWTON-RAPHSON ALGORITHM

DIVISION

The division operation can be written as:

$$N = D \cdot Q + R \qquad [1]$$

where N is dividend, D is divisor, Q is quotient, R is remainder and $|R| < |D|$.ulp and sign (R) = sign(N). The unit in the last position (ulp) represents the lowest term where ulp = 1 for integer numbers and ulp = $(\text{radix})^{-n}$ for n-bit fractional numbers [9].

The Newton-Raphson method is a well-known technique to find the root of nonlinear functions. This root can be calculated using an initial value by approaching the root quadratically [3][8]. The accuracy of the division operation doubles in each iteration. The initial value estimation can reduce the iteration number and increase accuracy.

$$X_{i+1} = X_i - \frac{f(X_i)}{f'(X_i)} \qquad [2]$$

The function $f(X) = D - \frac{1}{X}$ can be defined to calculate $X_{i+1} = \frac{1}{D}$ where D is the divisor. After applying $f(X)$ and $f'(X)$ to Equation [2], $X_{i+1} = \frac{1}{D}$ can be calculated as

$$X_{i+1} = X_i(2 - DX_i) \qquad [3]$$

After n[th] iteration, the value of X_{i+1} converges to 1/D and the quotient can be calculated as Q = N (1/D).

Design accuracy and error can be improved by increasing the number of iterations and better estimate of initial value [6]. The hardware implementation of Newton-Raphson division and its iteration steps are described in Figure 1 and Table I.

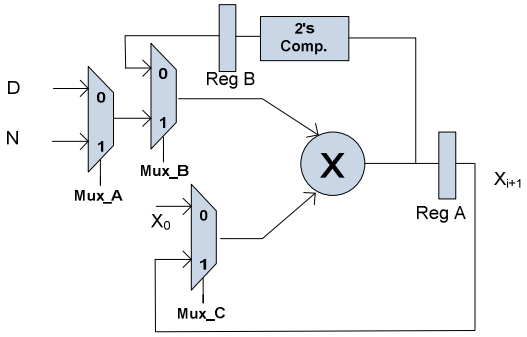

Figure 1 Newton-Raphson division method.

Table I. Newton-Raphson division cycles.

Operation Cycle	Mux A	Mux B	Mux C	Reg A	Reg B
1	0	1	0	---	2-D.X_0
2	0	0	0	X_1	---
3	0	1	1	---	2-D.X_1
4	0	0	1	X_2	---
5	0	1	1	---	2-D.X_2
6	0	0	1	X_3	---
7	0	1	1	N.X_3	---

Table I shows the operation of the division block. In cycle 1, pre-determined values of initial approximation X_0 and D are fetched into the multiplier. The product term is calculated and it's two's complement is stored in Reg B. In cycle 2, this complemented value is multiplied by X_0 and the result is stored in Reg A as X_1. After 3 iterations or 6 clock cycles, the value of X_3 converges to 1/D. After calculation of 1/D, the quotient can be calculated as shown in cycle 7 by multiplying X_3 by N.

INVERSE SQRT AND SQRT

Inverse sqrt operations can be generated using the Newton-Raphson algorithm. The function $f(X) = X^2 - \frac{1}{D}$ can be defined to calculate $X_{i+1} = \frac{1}{\sqrt{D}}$. After applying $f(X)$ and $f'(X)$ to Equation [2],

$$X_{i+1} = 2^{-1}X_i(3 - DX_i^2) \quad [4]$$

After n[th] iterations, the value of X_{i+1} will converge at the inverse sqrt of D. Calculations of the sqrt can be done by multiplying the final value of [4] by D.

The hardware implementation of Newton-Raphson sqrt and inverse sqrt methods and their iteration steps are described in Figure 2 and Table 2.

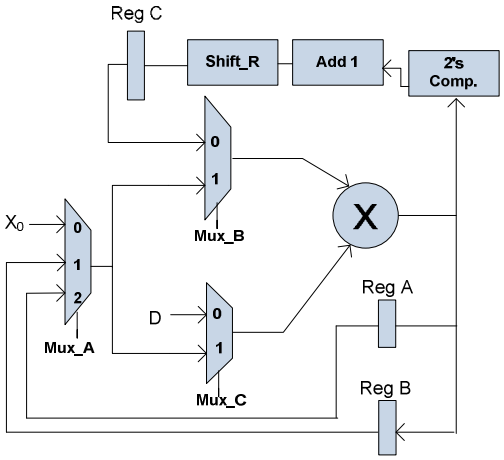

Figure 2 Newton-Raphson inverse-sqrt and sqrt methods.

Table II. Newton Raphson inverse sqrt and sqrt cycles.

Operation Cycle	Mux A	Mux B	Mux C	Reg A	Reg B	Reg C
1	0	1	1	X_0^2	----	----
2	2	1	0	DX_0^2	----	$2^{-1}(3 - DX_0^2)$
3	0	0	1	$2^{-1}X_0(3 - DX_0^2)$	$2^{-1}X_0(3 - DX_0^2)$	----
4	1	1	1	X_1^2	X_1	----
5	2	1	0	DX_1^2	----	$2^{-1}(3 - DX_1^2)$
6	1	0	1	$2^{-1}X_1(3 - DX_1^2)$	$2^{-1}X_1(3 - DX_1^2)$	----
7	1	1	1	X_2^2	X_2	----
8	2	1	0	DX_2^2	----	$2^{-1}(3 - DX_2^2)$
9	1	0	1	$2^{-1}X_2(3 - DX_2^2)$	$2^{-1}X_2(3 - DX_2^2)$	----
10	1	1	0	D.X_3	----	----

In Table II, operation of the inverse sqrt and sqrt blocks are shown. In cycle 1, X_0^2 is calculated. This value is stored in Reg A. In cycle 2, X_0^2 is multiplied by D and the result D X_0^2 is stored in Reg A and 2^{-1}(3 - D X_0^2) is stored in Reg C. In cycle 3, 2^{-1}(3 - D X_0^2) is multiplied by X_0 and result 2^{-1} X_0(3 - D X_0^2) is stored in Reg A and Reg B. This is the end of first iteration. The registers A and B are holding the value of X_1 that will be used in the second iteration. Iterations 2 and 3 are done in cycles 4 through 9, and the final result X_3 will converge into $\frac{1}{\sqrt{D}}$. In cycle 10, $\sqrt{D}$ is calculated by multiplying $\frac{1}{\sqrt{D}}D$.

B. GOLDSCHMIDT ALGORITHM

DIVISION

Similar to the Newton-Raphson method, the Goldschmidt iteration is also a quadratic convergence method that is used in many fast processors [4]. The division method that is described in the equation [1] will be used here to explain the Goldschmidt method. This method

will try to calculate $Q = \frac{N}{D}$ by using a sequence of F_1, F_2, F_3,...., the product of which goes to 1/D. Consequently, Q can be calculated as;

$$Q = N.F_1.F_2.F_3.F_4 \ldots\ldots \quad [5]$$

It is shown clearly in [5] that calculating $F_1.F_2.F_3.F_4$ ... will yield the solution of the division operation. In addition, the inverse operation $Inv = \frac{1}{D}$ can be calculated easily by making N=1 in the equation [5]. The inverse of any number other than zero can be calculated as;

$$Inv = \frac{1}{D} = F_1.F_2.F_3.F_4 \ldots\ldots \quad [6]$$

Calculating $F_1.F_2.F_3.F_4 \ldots\ldots$ can be done using initial values

$$X_0 = N \quad [7]$$

$$Y_0 = D$$

$$F_0 = \pm 0.75$$

and using the following iterative equations,

$$X_{i+1} = X_i.F_i \quad [8]$$

$$Y_{i+1} = Y_i.F_i$$

$$F_{i+1} = (2 - Y_{i+1})$$

After n[th] iteration, the value of X_{i+1} converges to the quotient (Q). This paper will use the fixed iteration method in a way similar to that used in the Newton-Raphson Method. The hardware implementation of the Goldschmidt division method and iteration steps are described in Figure 3 and Table III.

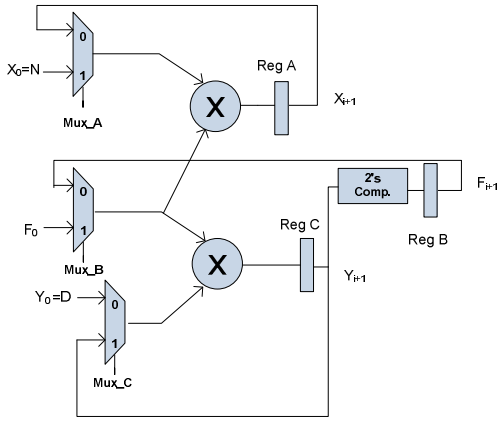

Figure 3 Goldschmidt division method with two multipliers.

Table III. Goldschmidt division cycles.

Cycle	Mux A	Mux B	Mux C	Reg A	Reg B	Reg C
0	1	1	0	---	----	----
1	0	0	1	X_1	F_1	Y_1
2	0	0	1	X_2	F_2	Y_2
3	0	0	1	X_3	F_3	Y_3

INVERSE SQRT AND SQRT

The Goldschmidt method can be used to calculate square-root $\left(S = \sqrt{D}\right)$ and inverse square-root $\left(IS = \frac{1}{\sqrt{D}}\right)$. The inverse square-root will be calculated, and with this value the sqrt can be calculated by multiplying $\left(\frac{1}{\sqrt{D}}\right)$ by D. Calculating the inverse sqrt using the Goldschmidt method also requires a new sequence of $F_0, F_1, F_2, F_3 \ldots\ldots$ and calculating this can be done using the initial values

$$X_0 = D \quad [9]$$

$$IS_0 = F_0 = 0.75$$

and using the following iterative equations,

$$X_{i+1} = X_i.{F_i}^2 \quad [10]$$

$$F_{i+1} = \frac{(3 - X_{i+1})}{2}$$

$$IS_{i+1} = IS_i.F_{i+1}$$

After the n[th] iteration, the value of IS_{i+1} converges to the inverse sqrt, and then the sqrt (S) can be calculated as,

$$S = D.IS_{i+1} = D.IS_i.F_{i+1} \quad [11]$$

The hardware implementation of the Goldschmidt sqrt and inverse sqrt methods and their iteration steps are described in Figure 4 and Table IV.

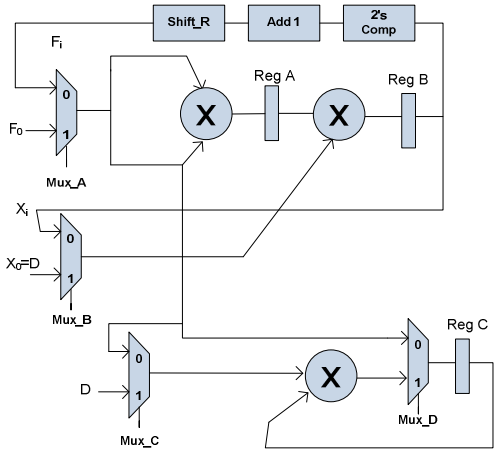

Figure 4 Goldschmidt inverse-sqrt and sqrt methods with three multipliers.

Table IV. Goldschmidt Inverse-sqrt and sqrt cycles.

Cycle	Mux A	Mux B	Mux C	Mux D	Reg A	Reg B	Reg C	Operation
0	1	1	0	0	F_0^2	$X_0F_0^2$	F_0	Inv Sqrt
1	0	0	0	1	F_1^2	$X_1F_1^2$	F_0F_1	Inv Sqrt
2	0	0	0	1	F_2^2	$X_2F_2^2$	$F_0F_1F_2$	Inv Sqrt
3	---	---	1	1	---	---	DF_0F_1F	Sqrt

III. UNIFIED DESIGN

The unified design calculates the division, sqrt, and inverse sqrt based on selection. The block diagram of this design is shown in Figure 5. This design has n-bit inputs A

and B and generates n-bit output Z. Based on the selection bits S_0S_1, these blocks operate as division, sqrt or inverse sqrt operations.

S_0S_1	Operation
00	Division
01	Square Root
10	Inverse SQ

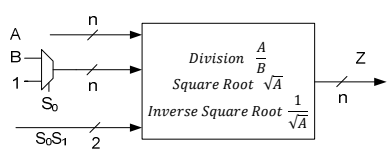

Figure 5 The unified division, sqrt and inverse sqrt and its operation.

The architecture of unified division, sqrt, and inverse sqrt design and its operation based on Newton-Raphson algorithm is shown in Figure 6 and Table V, respectively.

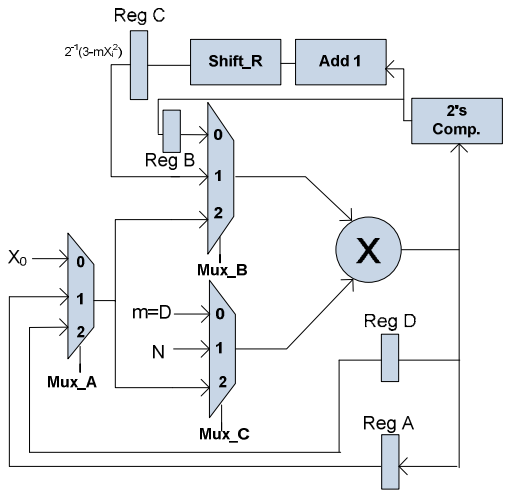

Figure 6 Unified Newton-Raphson division, inverse-sqrt and sqrt design.

Table V. Unified Newton-Raphson division, inverse-sqrt and sqrt design.

Cycle	Mux A	Mux B	Mux C	RegA	RegB	RegC	RegD	Op
0	0	2	0	mX_0	$2-mX_0$	----	----	Div
1	0	0	2	$X_0(2-mX_0)$	----	----	----	Div
0	0	2	2	X_0^2	----	----	----	I Sqrt
1	1	2	0	mX_0^2	----	$2^{-1}(3-mX_0^2)$	----	I Sqrt
2	0	1	2	$2^{-1}X_0(3-mX_0^2)$	----	----	X_1	I Sqrt
	2	2	0	$\sqrt{m}$	----	----	$\sqrt{m}$	Sqrt

The architecture of unified division, sqrt, and inverse sqrt design and its operation based on Goldschmidt algorithm is shown in Figure 7 and Table VI, respectively.

IV. The QR Transformation Using Householder Method

The QR matrix transformation is commonly used in DSP and wireless communication designs [10]. Using this transformation, a matrix **A** can be factored into **A** = **Q R** where **Q** is an orthogonal matrix, and where **R** is an upper right triangular matrix [2]. There are a few different techniques used to transform matrix **A**. The ones most commonly used are the Householder transformation, Givens Rotations [2][11] and Gram-Schmidt QR-Decomposition [12]. In this paper, the Householder implementation will be discussed, but this design can also be implemented into Givens Rotations. The QR factorization requires n-1 stages and these stages require multiplication, division and sqrt operations. The algorithm of QR factorization using the Householder method is given in Figure 8 and its block diagram in Figure 9.

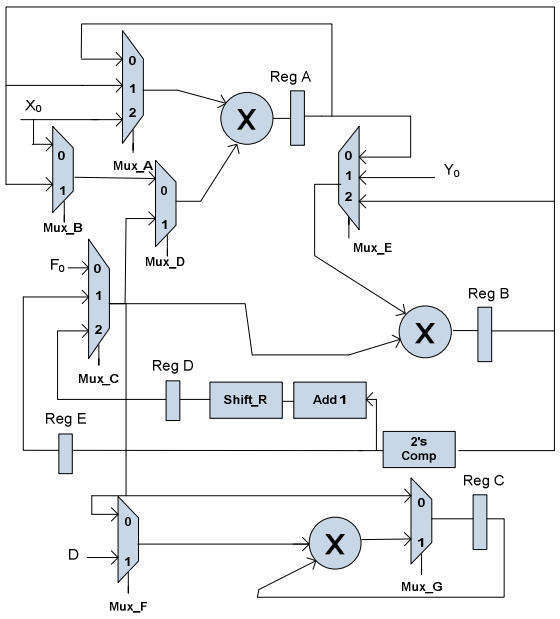

Figure 7 Unified Goldschmidt division, inverse-sqrt and sqrt design.

Table VI Unified Goldschmidt division, inverse-sqrt and sqrt design.

Cycle	MUX							Registers					Operation
	A	B	C	D	E	F	G	A	B	C	D	E	
0	2	---	0	1	1	---	---	X_1	Y_1	---	---	F_1	Div.
1	0	---	1	1	2	---	---	X_2	Y_2	---	----	F_2	Div.
0	2	0	0	0	0	---	0	X_0^2	$F_0X_0^2$	F_0	F_1	---	I.S
1	1	1	2	0	0	0	1	X_1^2	$F_1X_1^2$	F_0F_1	F_2	---	I.S
2	---	---	---	---	---	1	1	---	---	DF_0F_1	---	---	Sqrt

matrix **A**
$\mathbf{R_0} = \mathbf{A}$ //initilize
$\mathbf{Q_0} = \mathbf{I}$ //identity matrix
for i = 1 to n-1 // first column of A
  for j=1 to i-1 // initialize the first element
  a_j=0 // of k to zero
  end
  for j=i to n // from i^{th} element to n
  $\mathbf{a_j} = \mathbf{R_{j,i}}$
  end
$g = \sqrt{\sum_{j=0}^{n} \mathbf{a_j}}$ // calculate the norm(k)
$s = \sqrt{2g(g + |\mathbf{a_i}|)}$ // calculate s
if s ≠0 // all elements are zero if s=0
$\boldsymbol{w}_i = \frac{1}{s}(\mathbf{a_i} + \text{sign}(\mathbf{a_i})g)$ // calculate i^{th} w
$\mathbf{H_1}$ =I-2**ww'** // calculate H
$\mathbf{R_1} = \mathbf{H_1}\,\mathbf{R_0}$ // calculate R
$\mathbf{Q_1} = \mathbf{H_1}\mathbf{Q_0}$ // calculate Q
  End // repeat n-1times
end

Figure 8 . QR factorization algorithm.

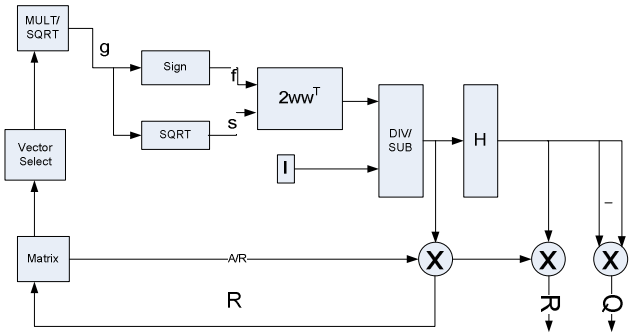

Figure 9 The QR Factorization block diagram.

The proposed design uses the same hardware block to calculate each step (total of n-1 for n x n matrix) to reduce the area of the overall design. Next, an n x m matrix A is used to show the calculation steps.

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,m} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,m} \end{bmatrix} \quad [12]$$

The first step is to calculate the first Householder transformation matrix H_1. In order to do this, the first column of matrix A needs to be used;

$$\mathrm{a} = \mathrm{A}(:,1) = \begin{bmatrix} a_{1,1} \\ a_{2,1} \\ \cdot \\ \cdot \\ a_{n,1} \end{bmatrix} \quad [13]$$

After that, norm (a) of this column matrix needs to be calculated;

$$g = \sum_{k=1}^{n} \mathrm{a}_{\mathrm{k},1}^2 = \sqrt{\mathrm{a}_{1,1}^2 + \mathrm{a}_{2,1}^2 + \cdots + \mathrm{a}_{\mathrm{n}-1,1}^2 + \mathrm{a}_{\mathrm{n},1}^2} \quad [14]$$

After calculating norm (a), the next value will be f;

$$f = sign(a_{1,1}) \times g \quad [15]$$

The next step is to calculate s;

$$s = \sqrt{2\mathrm{g}^2 + 2\mathrm{g}|a_{1,1}|} = \sqrt{2\mathrm{g}(\mathrm{g} + |a_{1,1}|)} \quad [16]$$

After calculating s and f, w can be calculated as;

$$w = \frac{1}{s} \times \begin{bmatrix} f_1 + a_{1,1} \\ a_{2,1} \\ \cdot \\ \cdot \\ a_{n,1} \end{bmatrix} \quad [17]$$

After w, the first H value can be calculated as;

$$H_1 = I - 2w_1 w_1^T \quad [18]$$

$$H_1 = I - \frac{2}{s^2} \times \begin{bmatrix} f_1 + a_{1,1} \\ a_{2,1} \\ \cdot \\ \cdot \\ a_{n,1} \end{bmatrix} \begin{bmatrix} f_1 + a_{1,1} & a_{2,1} & \ldots\ldots & a_{n,1} \end{bmatrix} \quad [19]$$

After this, the first value of R_1 can be found;

$$R_1 = H_1 \times A \quad [20]$$

After R_1 is found, the steps above will be repeated by replacing A with R_1 and R_2 and R_{n-1} and by shifting the columns to the right to find the final values for H_{n-1} and R_{n-1}. After these values are found, the QR transformation of matrix A can be accomplished by using the following;

$$R = R_{n-1} = H_{n-1} \times R_{n-2} \quad [21]$$

$$Q = H_1 \times H_2 \ldots \times H_{n-1} \quad [22]$$

Hardware realization of the QR algorithm using unified division, square-root and inverse square-root block is shown in Figure 11.

V. Results and Discussion

Implementation of the QR factorization of 8-bit, 2x2 matrix using VerilogHDL is shown in Figure 10. The area and clock speed of each module and top module is shown in Table 7. Synthesis is done using a Xilinx Spartan 3E board which contains 500,000 gates. The unified design reduces the area and uses only 5.8 percent of total Spartan 3E resources. This design uses less area and power consumption due to hardware unification. The low area and power consumption makes this design suitable for low cost FPGAs and low power VLSI applications. To make this design desirable for high speed low-power applications, matrix multiplication can be improved. This design uses a typical matrix multiplication for n x n matrix that has a computation time of $O(n^3)$ [13]. The computation can be decreased by using a fast matrix multiplication algorithms such as the Strassen algorithm that takes $O(n^{2.81})$ [13] or Coppersmith and Winograd's algorithm that takes $O(n^{2.38})$ [13]. The computation speed can also be increased by pipelining.

VI. Conclusion

In this paper, an area-efficient low power implementation of QR algorithm using Householder method is presented. To reduce the area and improve the speed, a unified design based on two different types of division, inverse sqrt and sqrt methods are used. This unified design can operate as a multiplication, addition, division, sqrt, inverse sqrt, and the QR factorization block by changing the controller ROM. Future work will focus on extending this unified design into different data sizes such as 16-bit and 32-bit to increase precision. Furthermore, an important extension of this work is to design a reconfigurable QR factorization of n x n matrix.

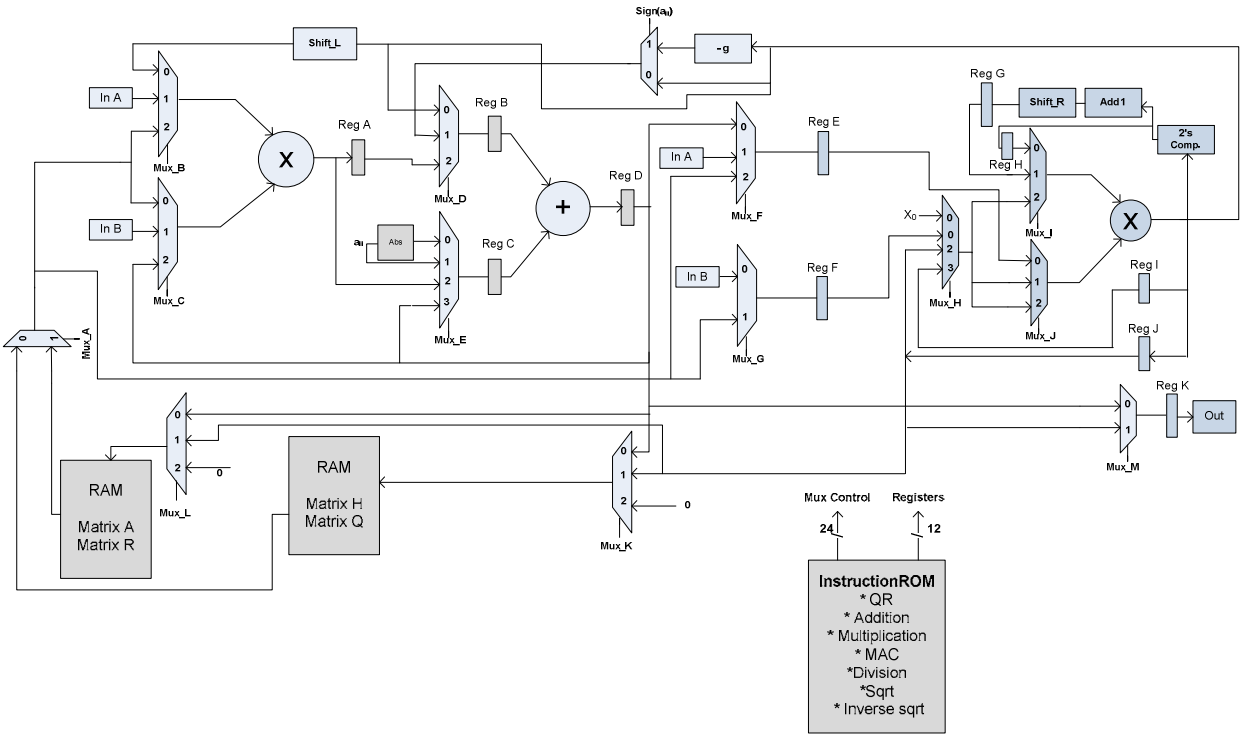

Figure 10. Hardware realization of QR Factorization using unified division, square-root and inverse square-root block.

Table VII. Results

Design	Newton-Raphson Based					Goldschmidt Based				
	Slice	LUT	FF	Speed (MHz)		Slice	LUT	FF	Speed (MHz)	
Division	84	167	32	68.2		162	324	38	74.8	
Sqrt	94	188	38	62.8		174	348	38	71.2	
Inverse Sqrt	94	188	38	62.8		174	348	38	71.2	
Unified Design	148	296	38	59.6		248	496	38	66.1	
MAC	80	160	32	224.12		80	160	32	224.12	
RAM 1	16	32	---	---		16	32	14	---	
RAM 2	16	32	---	---		16	32	14	---	
ROM	12	24	---	---		12	24	32	---	
QR*	**272**	**544**	**114**	**224**	**59**	**372**	**744**	**114**	**224**	**66.1**

* This design uses two clocks to optimize the speed. One clock speed is 224 MHz and the other 59 MHz. Final Q and R matrixes are calculated after 82 clock cycles as described in the instruction ROM .

REFERENCES

[1] C.K. Singh, S.H. Prasad, P.P. Balsara, "A Fixed-Point Implementation for QR Decomposition", *IEEE Workshop on Circuits and Systems*, Pages 75-78, Oct. 2006.

[2] L.V. Fausett, "*Numerical Methods, Algorithms and Applications"*. New Jersey: Ch 4, pp 150-162, Prentice Hall, 2003.

[3] S. F. Oberman, M. J. Flynn, "Division Algorithms and Implementations". *IEEE Transactions on Computers*, Vol. 46, No. 8, August 1997.

[4] P. Markstein, "Software Division and Square Root Using Goldschmidt's Algorithms", *Hewlett-Packard Laboratories,* Pages 146-157.

[5] M. D. Ercegovac, D. W. Matula, J.M. Muller. "Improving Goldschmidt Division, Square Root, and Square Root Reciprocal". *IEEE Transactions on Computers*, Vol. 49, No. 7, July 2000.

[6] M. J. Schulte, J. Omar, E. E. Swartzlander Jr., "Optimal Initial Approximations for the Newton-Raphson Division Algorithm". *Springer Journal,* June 1994.

[7] G. Even, P. Michael Serdel, W. E. Ferguson, "A Parametric Error Analysis of Goldschmidt's Division Algorithm". *IEEE Symposium on Computer Arithmetic*, 2003.

[8] W.L.Gallagher, E.E. Swartzlander Jr.. "Fault-Tolerant Newton-Raphson and Goldschmidt Dividers Using Time Shared TMR". *IEEE Transactions on Computers*, pages588-595, June 2000.

[9] M. D. Ercegovac, T. Lang, "*Digital Arithmetic"*. San Francisco: Ch 5 , pp 248-250, Morgan Kaufmann, 2004.

[10] M. Misra, R. Moona, "Design of Systolic arrays for QR Decomposition", *International Conference on Computer System and Education,* IISc, 1994

[11] Y. Tai, C. D. Lo, K. Psarris "Applying Out-Of-Core QR Decomposition Algorithms on FPGA Based Systems". *IEEE Field Programmable Logic and Applications*, pages 86-91, Aug 2007.

[12] P. N. Ganchosov, G.K. Kuzmanov, H. Kabakchiev, V. Behar, R. P. Romansky, G. N. Gaydadjiev, "FPGA Implementation of Modified Gram-Schmidt QR-Decomposition", *Proceedings* of the 3rd HiPEAC Workshop on Reconfigurable Computing, pp. 41-51, January 2009.

[13] S. Robinson "Toward an Optimal Algorithm for Matrix Multiplication" . *SIAM News,* Volume 38, Number 9, November 2005.