LOW LATENCY DIGIT-RECURRENCE RECIPROCAL AND SQUARE-ROOT RECIPROCAL ALGORITHM AND ARCHITECTURE

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

Master of Technology
In
VLSI Design and Embedded Systems

By

MADHU BABU TIRIVEEDHI

Roll No: 20607004

Department of Electronics and Communication Engineering
National Institute of Technology
Rourkela
2006-2008

LOW LATENCY DIGIT-RECURRENCE RECIPROCAL AND SQUARE-ROOT RECIPROCAL ALGORITHM AND ARCHITECTURE

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

Master of Technology
In
VLSI Design and Embedded Systems

By

MADHU BABU TIRIVEEDHI

Roll no: 20607004

Under the Guidance of

Prof. G.S. RATH

Department of Electronics and Communication Engineering
National Institute of Technology
Rourkela
2006-2008

National Institute of Technology

Rourkela

CERTIFICATE

This is to certify that the thesis entitled. "**LOW LATENCY DIGIT-RECURRENCE RECIPROCAL AND SQUARE-ROOT RECIPROCAL ALGORITHM AND ARCHITECTURE**" submitted by Mr. MADHU BABU TIRIVEEDHI in partial fulfillment of the requirements for the award of Master of Technology Degree in Electronics and Communication Engineering with specialization in "**VLSI Design and Embedded System**" at the National Institute of Technology, Rourkela (Deemed University) is an authentic work carried out by his under my supervision and guidance.

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

Date:

Prof. G.S. RATH

Dept. of Electronics and Communication Engg.

National Institute of Technology

Rourkela-769008

ACKNOWLEDGEMENTS

This project is by far the most significant accomplishment in my life and it would be impossible without people who supported me and believed in me.

I would like to extend my gratitude and my sincere thanks to my honorable, esteemed supervisor **Prof. G.S. RATH,** Department of Electronics and Communication Engineering. He is not only a great lecturer with deep vision but also and most importantly a kind person. I sincerely thank for his exemplary guidance and encouragement. His trust and support inspired me in the most important moments of making right decisions and I am glad to work with him.

I want to thank all my teachers **Prof. G.Panda, Prof. K. K. Mahapatra, Prof. S.K. Patra, Prof. S. Meher** for providing a solid background for my studies and research thereafter. They have been great sources of inspiration to me and I thank them from the bottom of my heart.

I would like to thank all my friends and especially my classmates for all the thoughtful and mind stimulating discussions we had, which prompted us to think beyond the obvious. I've enjoyed their companionship so much during my stay at NIT, Rourkela.

I would like to thank all those who made my stay in Rourkela an unforgettable and rewarding experience.

Last but not least I would like to thank my parents, who taught me the value of hard work by their own example. They rendered me enormous support during the whole tenure of my stay in NIT Rourkela.

MADHU BABU TIRIVEEDHI

ABSTRACT

The reciprocal and square root reciprocal operations are important in several applications such as computer graphics and scientific computation. For the two operations, an algorithm that combines a digit-by-digit module and one iteration of the Newton-Raphson approximation is used. The latter is implemented by a digit-recurrence, which uses the digits produced by the digit-by-digit part. In this way, both parts execute in an overlapped manner, so that the total number of cycles is about half of the number that would be required by the digit-by-digit part alone. Since the approximation does not produce correct rounding in a few cases, for applications where exact rounding is required, the result is only computed by the digit-by-digit module.

Radix-4 implementations for combined unit are described. The result of the evaluation shows that the cycle time is the same as that of the digit-by-digit unit and that, as a consequence, the execution time is almost halved. Because of the approximation part, the area almost doubles of the digit-by-digit area.

In this Dissertation, an implementation algorithm and architecture for low latency digit recurrence reciprocal and squareroot reciprocal has been presented. The implementation is based on Radix-4, considering execution time and system complexity. This project aims to provide modules written in the Very High Speed Integrated Circuit Hardware Description Language (VHDL) that can be used to model low latency digit recurrence reciprocal and square root reciprocal architecture.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Digital arithmetic operations are very important in the design of digital processors and application-specific systems. Among the arithmetic operations, the reciprocal and square root reciprocal operations are widely used in applications such as graphics and scientific computation. Therefore, different kind of schemes has been developed. There are several algorithms for the computation of reciprocal and square root reciprocal. Conventional algorithms include: 1) Digit-by-digit algorithms. 2) Quadratic convergent (Newton-Raphson) algorithms. 3) Polynomial approximations.

Digit-by-digit algorithms are based on a digit recurrence. In this method, the quotient of reciprocal computation is represented in a radix-r form and one digit of it is obtained per iteration. The digit recurrence involves a digit selection, a digit multiplication and an addition. The Newton-Raphson algorithms and polynomial approximations compute the reciprocal by iteratively improving an initial approximation. The most complex operation involved in the iteration is multiplication.

Newton-Raphson method and polynomial approximations have a quadratic convergence rate, which means that the number of bits of accuracy of the approximation doubles after each iteration, therefore, reduce the number of iterations. However, both Newton-Raphson method and polynomial approximations method require a dedicated parallel multiplier and additional tables. In contrast, digit-by-digit method has the advantage of low hardware overhead, but liner convergence
With more iterations.

In this project, new algorithms are used, which combine a digit-by-digit part and one iteration of a quadratic convergence approximation. The latter uses the digits produced by the former part, so that two parts can execute in parallel fashion, thus only half of the iterations are needed as compared to the digit-by-digit part alone. The execution time is reduced by implementing the approximation as digit-recurrence which is performed in an overlapped manner with the digit-by-digit part. This requires two data paths operating in parallel.

Radix-4 implementations have been done. This is based on the consideration of system complexity and execution time. High radix makes the implementation very complex. Since reciprocal and square root reciprocal operations have similar characteristics, it is considered advantageous to have a single scheme to implement both functions. The combined unit for both reciprocal and square root reciprocal are presented. This combined implementation needs a single selection function.

The comparison has been performed among different algorithms and different radix. The results show that the proposed new schemes have nearly the same cycle time as the corresponding digit-by-digit schemes. But the number of iterations is only half of the digit-by-digit schemes alone. The total delay is reduced roughly by half. From the evaluation, it can be concluded that the new algorithm implementation is a low latency solution with moderate hardware overhead.

The remainder of the paper is organized as following:

In Chapter 2, the floating point number system using the IEEE-754 standard is introduced at the beginning, in order to give a background theory. Then the algorithms for computation of reciprocal and square root reciprocal are described. The conventional digit-by-digit algorithm and Newton- Raphson approximation are introduced respectively, then the new algorithm which combined the digit-recurrence and approximation together is described. The on-the-fly conversion and rounding algorithms are introduced as well, which is an important part for the combined unit implementation

.

Chapter 3 presented the full architecture design. It started with the reciprocal architecture design, then the square root architecture design and finally, the combined unit architecture design. The quotient digit selection function is introduced in details in this chapter.

Chapter 4 described the procedure of implementation and results. The critical path and power dissipation are estimated. The results are analyzed and evaluated. Chapter 5 gave a conclusion.

2. ALGORITHM

This chapter gives a detailed description of digit-by-digit algorithm, Newton-Raphson approximation algorithm and new algorithm used in this work. The Floating Point Standard 754 is introduced firstly to provide a background theory.

2.1 IEEE Floating Point Standard 754:

Since reciprocal and square root reciprocal computation require real numbers, the floating point representation is used in the design and implementation.

Parameters for representation:

There are several parameters that define a floating point representation system.
1. Sign S: One bit. S=1 if the number is negative

2. Magnitude (also called the significand): Represented in radix 2 with one integer bit.

$$1.\,\mathrm{F}$$

Where F is fraction part with f bits and the most-significant 1 is the hidden bit.
The range of the (normalized) significand

$$1 \le 1.F \le 2 - 2^{-f}$$

3. Exponent: base 2 and biased representation. The exponent field e, biased with bias

$$B = 2^{e-1} - 1$$

IEEE double precision standard:

The double precision floating point number has 64 bits.

S	EEEEEEEEEEE	MMMMMM	...	MMMMMMMM
63	6252	51		0

1. First bit is the sign bit S.

2. Next 11 bits are the exponent bits E.

3. Final 52 bits are the significand, or magnitude M.

$$V = (-1)^S (1.M) 2^{E-1023} \qquad \text{With} \quad 0<E<2047$$

4. Representation is normalized in 1.0 < 1. M < 2.0. Normally, “one” (1.M) is implicit in representation.

5. Exponent is biased (e.g. exp = 0 then E = 0 + 1023).

For single precision floating point representation (32 bits), S is 1 bit, E is 8 bits and F is 23 bits.

In this work, we use double precision floating point representation. During the implementation, the operands and result are in sign-and-magnitude representation. Only magnitudes are considered.

2.2 Digit-by-Digit Algorithm:

The digit-by-digit algorithm is developed for division operation. Naturally, it applies to the reciprocal and is the basis for square root reciprocal operation. The algorithm is based on digit recurrence method. In this method, the quotient is represented in a radix-r form and one digit of it is obtained per iteration. The digit-recurrence involves a digit selection, a digit multiplication, and an addition.

Operand and Result

The reciprocal operation is defined as

$$q=1/d$$

Where dividend 1 and the divisor d are the operands and the result is the quotient q. The operands and result are represented in double precision floating-point using the IEEE-754 standard, namely, with significand in the range [2,1) with a precision of 53 bits. The range of the quotient is

$$1 \le q < 2$$

Corresponding to

$$1/2 \le d < 1$$

To achieve the result in the specified range, the input significand d may be shifted within the interval [1/2,1). Correspondingly, the exponent has to be adjusted.

Define square root reciprocal as

$$p = 1/\sqrt{d}$$

Again, the range of the result

$$1 \le p < 2$$

Correspondingly, the range of the operand

$$1/4 < d < 1$$

Digit-by-digit recurrence for reciprocal:

The digit-by-digit algorithm consists of n iterations of a recurrence, in which each iteration produces one digit of the quotient. The value of the quotient after j iterations Q[j] is defined as

$$Q[J] = Q[0] + \sum_{i=1}^{j} q_i r^{-i} \quad (1)$$

Where r is radix, Q[0] is determined by the initialization.

The residual (or partial remainder) w define as

$$W[J] = r^j (1 - dQ[J]) \quad (2)$$

The expression for recurrence is

$$w[j+1] = rw[j] - q_{j+1}d \qquad (3)$$

$$q_{j+2} = SEL(rw[j+1), d) \qquad (4)$$

Expression (4) is the quotient-digit-selection function. It is used to select a suitable value of qj+1. The digit qj+1 take values in the set

$$q_{j+1}\varepsilon\{-a, \ldots . -1, 0, 1, \ldots . a\}$$

$$\rho = a / r - 1$$

Where ρ is the redundancy factor

The digit q_{j+1} is selected so that w [j+1]is bounded by

$$-\rho d \le w[j] \le \rho d \ .$$

Quotient-digit-selection function scheme is shown in Figure 1.

Before recurrence, the initialization step is needed to initialize w [0] and Q [0] in order to assure convergence. Initialization will be introduced in chapter 3.

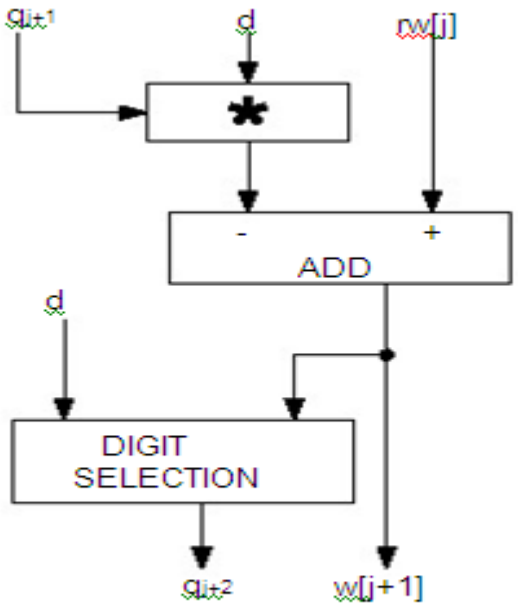

Figure 1 Scheme of digit-by-digit recurrence for reciprocal

Digit-by-digit recurrence for Square Root Reciprocal:

For the square root reciprocal recurrence, the partial result up to iteration j, P[j] defines as

$$p[j] = p[0] + \sum_{i=1}^{j} p_i r^{-i} \qquad (5)$$

Where P [0] is initialization

The residual is defined as

$$w[j] = (1/2) r^{j} (1 - dp[j]^2) \qquad (6)$$

To simplify the description and the implementation, two variables are introduced:

$$D[j] = dp[j] \qquad (7)$$

$$C[j] = (1/2) r_{-(j+1)} d \qquad (8)$$

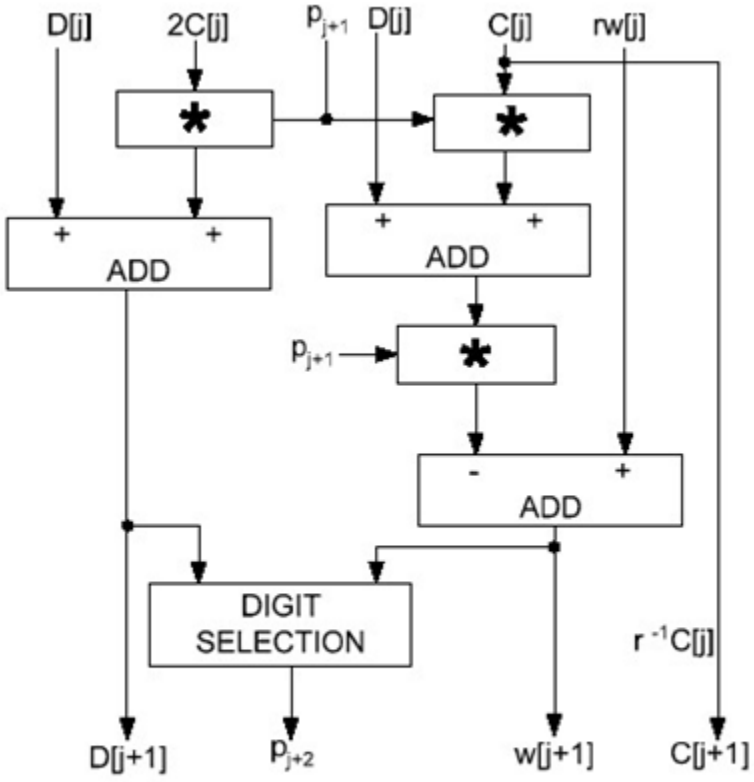

Figure 2 Scheme of digit-by-digit recurrence for square root reciprocal

Using these variables, the recurrence defined as

$$w[j+1] = rw[j] - p_{j+1}D[j] - p^2{}_{j+1}C[j] \quad (9)$$

$$D[j+1] = D[j] + 2p_{j+1}C[j] \quad (10)$$

$$C[j+1] = r^{-1}C[j] \quad (11)$$

$$p_{j+2} = SEL(rw[j+1], D[j+1]) \quad (12)$$

It is necessary to initialize W [0], D [0], C[0] and P[0] before recurrence,.

The digit selection function is developed in [3]. The scheme is shown in Figure 2.

2.3 Newton-Raphson Approximation:

The Newton-Raphson method computes a function by iteratively improving an initial approximation. The most complex operation involved in the iteration is multiplication. This method has a quadratic convergence rate, which means that the number of bits of accuracy of the approximation doubles after each iteration. As a consequence, the number of iterations for a desired accuracy is smaller than for linear convergence (digit-by-digit). However, since full precision multiplications are involved, the delay of an iteration is larger.

Newton-Rahson method for reciprocal approximation:

The Newton-Raphson method illustrated here is the computation of the reciprocal function, since it is the basis for square root reciprocal.

The approximation is based on a general method to obtain the zero of a function. That is, the value of x for which f(x) = 0. If x[j] is an approximation of the zero, then a better approximation is

$$x[j+1] = x[j] - f(x[j]) / f\,'(x[j]) \qquad (13)$$

Where f ’(x[j]) is the derivative of f(x) with respect to x, evaluated at x[j].

For the approximation of reciprocal,

$$f(R) = 1/R - d \text{ (whose zero is } 1/d)$$

$$f\,'(R) = -1/R^2$$

Apply to expression (13), the recurrence is obtained.

$$R[j+1] = R[j](2 - R[j]d) \qquad (14)$$

The recurrence is initiated with an initial approximation R[0]. Each iteration requires two Multiplications and one subtraction from the value 2. The scheme is shown in Figure 3.

The convergence of this method is quadratic, that is, if the error at step j is E[j], then the error at step j+1 is E[j]2 . This can be shown as follows. Since R[j] is an approximation of 1/d, the relative error is

E[j] =1 – d R[j]

Then from expression (14)

$$R[j+1] = (1 - E[j]^2) / d$$

$$E[j+1] = 1 - dR[j+1] = E[j]^2 \qquad (15)$$

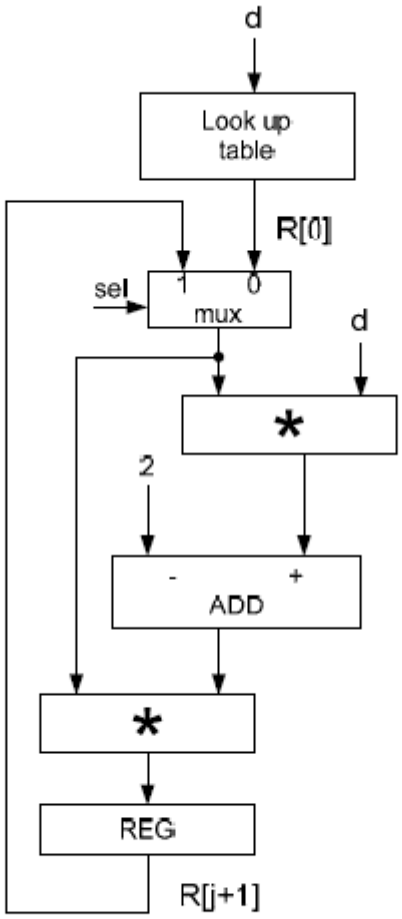

Figure 3 Scheme of Newton-Rahson method for reciprocal approximation

2.4 Proposed Algorithm:

The new algorithm used in this work consists of a digit-by-digit part followed by a Newton-Raphson approximation. Two parts operate in an overlapped manner to reduce the total delay. The scheme of new algorithm is illustrated in Figure 4. The algorithm consists two steps:

1. Module A (Digit-by-digit) obtains the approximation k using a digit recurrence and performing g

   Iterations, roughly half of final required precision

2. Module B (Newton-Raphson method) performs a better approximation by means of a digit recurrence.

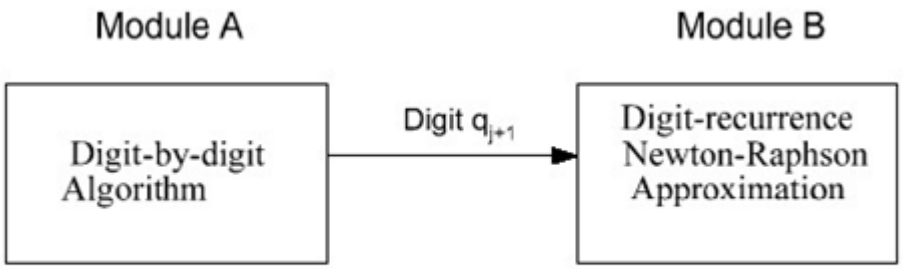

Figure 4 New algorithm scheme

This section introduces the approximation part.

Reciprocal approximation:

The digit-by-digit part produces an approximation k (k=Q[g]), following by one Newton-Raphson iteration. As described by the Newton-Raphson method, a better approximation is given by

$$A = k(2 - k\,d) \qquad (16)$$

From expression (15), we know that the convergence of this iteration is quadratic, that is, if the error of k is δ, then the error of A is E = δ^2 . This can be shown as follows. Since k is an approximation of 1/d, the relative error is

δ = 1- d k, and k = (1 - δ) / d

Applies to expression (16), then

$$A = (1-\delta^2)/d$$

The relative error of A is

$$E = 1 - dA = \delta^2$$

The approximation A is computed by means of a digit-recurrence to avoid multipliers and to speed up the computation. This approximation recurrence is overlapped with the computation of k. From expression (16), the approximation recurrence can be defined as

A[j] = Q[j](2 - d Q[j]) (17)

To eliminate variable shifts, define

$$E[j] = r^{2j} A[j]$$

Then

$$E[j+1] - r^2 E[j] = r^{2(j+1)}(Q[j+1](2 - dQ[j+1] - Q[j](2 - dQ[j]))$$

$$E[j+1] - r^2 E[j] = r^{2(j+1)}((Q[j] + q_{j+1} r^{-(j+1)})(2 - dQ[j+1]) - Q[j](2 - dQ[j]))$$

Which simplifies to

$$E[j+1] - r^2 E[j] = q_{j+1} r^{j+1}(2 - dQ[j+1] - dQ[j])$$

As shown of expression (2),

$$w[j] = r^j (1 - dQ[j])$$

$$Q[j+1] = Q[j] + q_{j+1} r^{-(j+1)}$$

The expression for approximation recurrence is obtained as

$$E[j+1] = r^2 E[j] + q_{j+1}(2rw[j] - q_{j+1} d) \qquad (18)$$

Where w[j] and - qj+1 d are computed by the digit-by-digit recurrences.

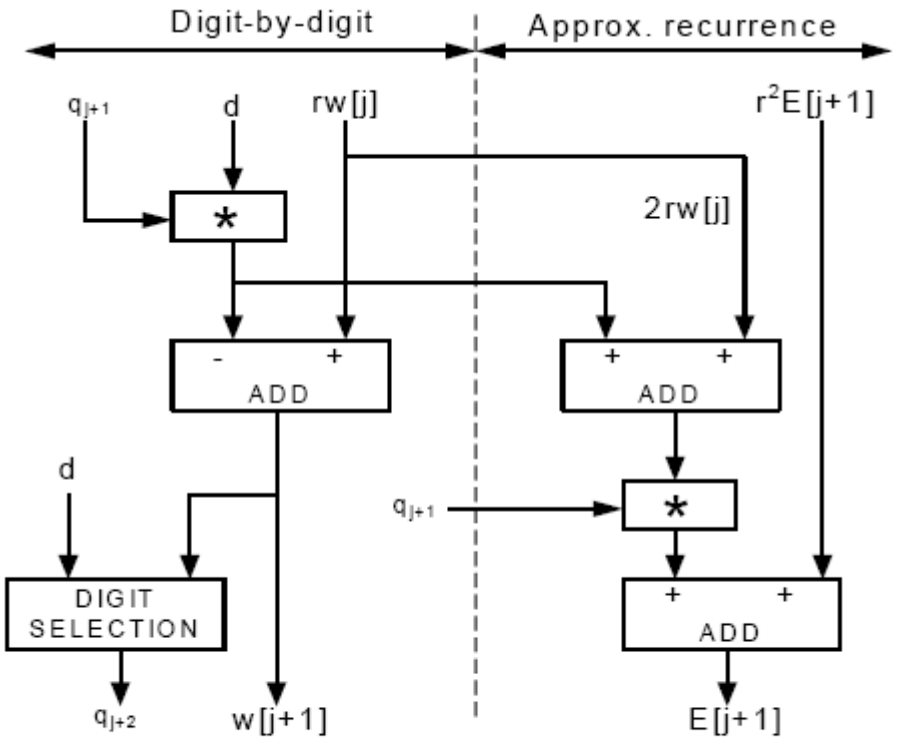

Figure 5 Scheme for reciprocal algorithm

Figure 5 shows the scheme of reciprocal algorithm. The digit-by-digit recurrence and approximation recurrence are in overlapped manner. After g iterations, g is roughly half of the iterations as compared to use the digit-by-digit algorithm alone, the final result E[g] obtained.

Square Root Reciprocal approximation:

Similarly to the case for reciprocal, the square root reciprocal approximation is defined as

$$B = k(3/2 - k^2 d/2) \qquad (19)$$

Again, in this case the convergence is quadratic. If δ and E are the relative errors of k and B respectively, then

$$\delta = 1 - k\sqrt{d} \quad \text{and} \quad E = 1 - B\sqrt{d}$$

Therefore,

$$k\sqrt{d} = 1 - \delta \quad \text{and} \quad E = 1 - (k\sqrt{d}/2)(3 - dk^2)$$

Consequently,

$$E = 1 - 1/2(1-\delta)(3-(1-\delta)^2) = (\delta^2/2)(3-\delta)$$

Then the relative error of B has a complexity of $Q(\delta^2)$

Since k = P[g] (g is the total number of iterations), similar to the case for reciprocal, we define

$$H[j] = r^{2j} p[j](3/2 - dp[j]^2/2) = r^{2j} p[j](1 + r^{-j} w[j])$$

So that $B = r^{-2g} p[g]$.

The recurrence for H[j] is obtained by

$$H[j+1] = r^2 H[j] + r^{2(j+1)}[(p[j] + p_{j+1} r^{-(j+1)}).(1 + r^{-(j+1)} w[j+1]) - p[j](1 + r^{-j} w[j])$$

Using the recurrence given in (9) results in

$$H[j+1] = r^2 H[j] + p_{j+1}(2rw[j] + w[j+1] - p_{j+1} D[j]/2) \qquad (20)$$

The initial condition is

$$H[0] = p[0](3/2 - dp^2[0]/2)$$

The scheme of square root reciprocal algorithm is shown in Figure 6.

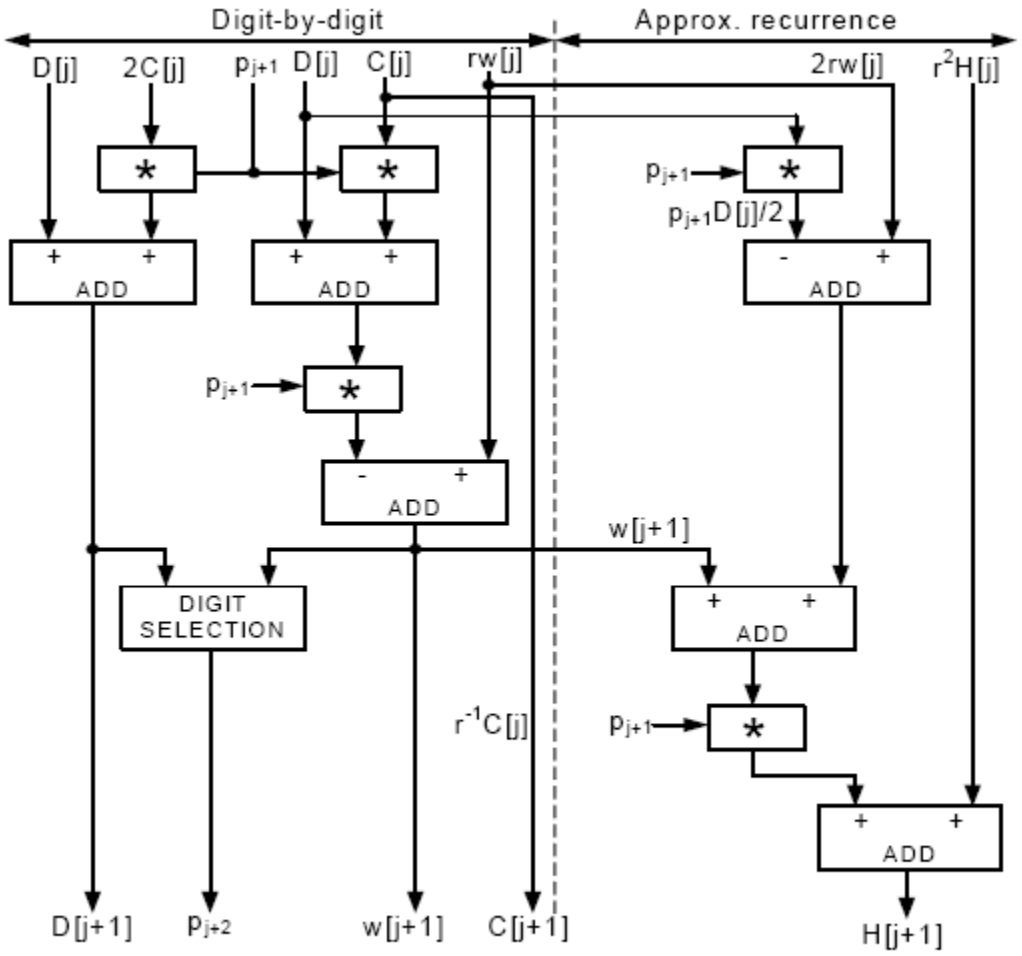

Figure 6 Scheme for square root reciprocal algorithm

2.5 On-the-fly Conversion and Rounding:

The on-the-fly conversion is to convert quotient from signed-digit representation to conventional representation. A simple alternative is to do an addition step after the quotient is completely computed. However, this addition would increase the delay. The on-the-fly conversion algorithm performs the conversion as the digits are produced. The on-the-fly conversion algorithm for digit- by-digit part can be expressed as following

:

Let Q[j] be the digit vector of the converted quotient consisting of the j most-significant digits, that is,

$$Q[j] = \sum_{i=1}^{j} q_i r^{-i}$$

Then obtain,

$$Q[j+1] = Q[j] + q_{j+1} r^{-(j+1)}$$

Since q_{j+1} can be negative, express algorithm as

$$Q[j+1] = Q[j] + q_{i+1} r^{-(i+1)} \qquad \text{If} \quad q_{i+1} \geq 0$$

$$Q[j+1] = Q[j] - r^{-i} + (r - |q_{i+1}|) r^{-(i+1)} \qquad \text{If } q_{i+1} < 0$$

This algorithm has the disadvantage that the subtraction $Q[j] - r^{-j}$ requires the propagation of a borrow and, therefore, is slow. To avoid this propagation, another form is defined as

$$QM[j] = Q[j] - r^{-i}$$

Using this form, the conversion algorithm becomes

$$Q[j+1] = (Q[j], q_{i+1}) \qquad \text{If} \quad q_{i+1} \geq 0$$

$$Q[j+1] = (QM[j], (r - |q_{i+1}|)) \qquad \text{If} \quad q_{i+1} < 0$$

$$QM[j+1] = (Q[j], q_{i+1} - 1) \qquad \text{If} \quad q_{i+1} < 0$$

$$QM[j+1] = (QM[j], ((r-1) - |q_{i+1}|)) \qquad \text{If} \quad q_{i+1} \leq 0$$

So that the subtraction is replaced by loading the form QM[j], then no carry/borrow is propagated. The form QM[j] also needs to be updated as shown in above expression. An example is given in Figure 7 to illustrate the on-the-fly conversion algorithm.

j	q_j	Q	QM
1	1	x x x x x x x x 1	x x x x x x x x 0
2	2	x x x x x x x 1 2	x x x x x x x 1 1
3	0	x x x x x x 1 2 0	x x x x x x 1 1 3
4	-1	x x x x x 1 1 3 3	x x x x x 1 1 3 2
5	0	x x x x 1 1 3 3 0	x x x x 1 1 3 2 3
6	0	x x x 1 1 3 3 0 0	x x x 1 1 3 2 3 3
7	2	x x 1 1 3 3 0 0 2	x x 1 1 3 3 0 0 1
8	-2	x 1 1 3 3 0 0 1 2	x 1 1 3 3 0 0 1 1
9	-1	1 1 3 3 0 0 1 1 3	1 1 3 3 0 0 1 1 2

Figure 7 Example of radix-4 on-the-fly conversion

To perform the on-the-fly conversion and round, three registers are needed to store Q, QM, QP as shown in Figure 8, However, when the rounding is done in the least-significant position, and a< r -1, two registers Q and QM are sufficient. These registers are shifted one digit left each cycle, while new least significant digits are inserted into registers, depending on the value of qj . Registers Q and QM are updated each iteration by the following rules:

$$Q[j] \le (shl(Q[j-1], q_j) \qquad q_j > 0$$
$$QM[j] <= (shl(Q[j-1]), q_j - 1)$$

$$Q[j] <= (shl(Q[j-1]), 0) \qquad q_i = 0$$
$$QM[j] <= (shl(QM[j-1], (r-1)$$

$$Q[j] <= (shl(QM[j-1], r - |q_j|) \qquad q_i < 0$$
$$QM[j] <= (shl(QM[j-1]), (r-1) - |q_j|)$$

For example, $Q[j] <= (shl(Q[j-1]), q_j)$ means that the register Q at iteration j is shifted one digit left and the last digit is loaded with q_j. In QM, the current digit is given by $q_j - 1$ (Mod r).

Register QP is updated by following rules,

$$QP[j] <= (shl(QP[j-1], 0) \quad \text{If} \quad q_j = r-1$$

$$QP[j] <= (shl(Q[j-1], q_j + 1) \quad \text{If} \quad -1 \le q_j \le r-2$$

$$QP[j] <= (shl(QM[j-1], r - |q_j| + 1) \quad \text{If} \quad q_j < -1$$

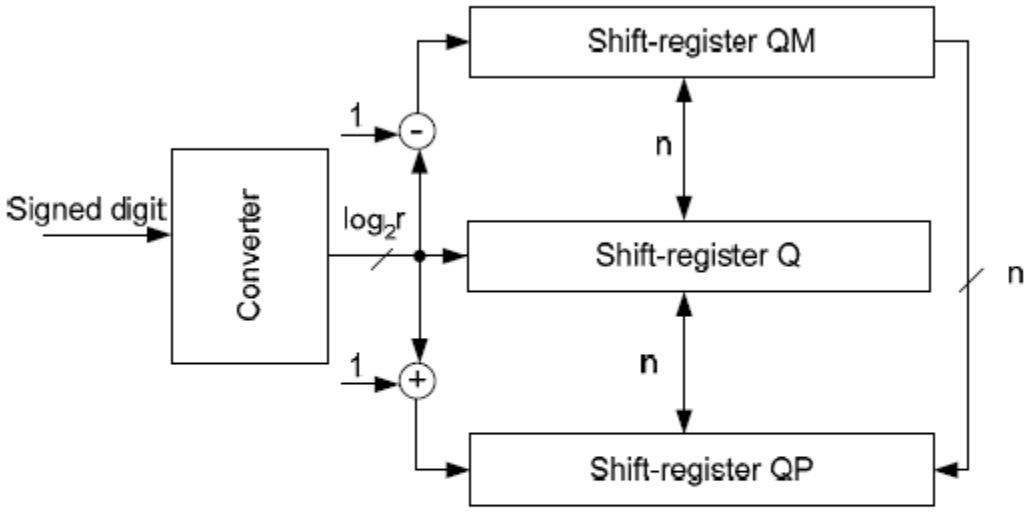

Figure 8 On-the-fly conversion and round unit

Table 1 Value of p in the rounding step

q_m	SIGN	ZERO	p	case
r - 1	0	0	0	1
	0	1	0	1
	1	-	r - 1	2
$0 \le q_m \le r-1$	0	0	$g_m + 1$	2
	0	1	g_m + G1	2
	1	-	g_m	2
-1	0	-	0	2
	1	-	r - 1	3
$q_m < -1$	0	0	$g_m + 1$	3
	0	1	g_m + G1	3
	1	-	g_m	3

The rounding is performed in the last cycle. First the quotient digit qm is converted into gm = qm (mod r). Then the rounded digit p is computed according to Table 1, where SIGN = 0 if the final residual is positive, ZERO = 1 if it is zero, and G1(guard bit) represents the bit before the least significant in gm. Two operations are performed in the rounding step:

1. If the remainder is negative, the quotient must be decremented by 1 (in rounding position).

2. To round-to-the-nearest 1 has always to be added in rounding position.

Finally, the quotient is obtained by

$$q = shl(QP[m-1], p_t) \qquad \text{if case 1}$$

$$q = shl(Q[m-1], p_t) \qquad \text{if case 2}$$

$$q = shl(QM[m-1], p_t) \qquad \text{if case 3}$$

Where . $p_t = [p/2]$

On-the-fly conversion for Newton-Raphson approximation part is similar to digit-by-digit part. Each iteration produces one digit (Radix-16) and store in the register. The register is shifted one digit left with insertion of new digit at least-significant position. Since approximation E[j] (or H[j] for combined unit) is in carry-save form, a short addition is needed to convert produced digit to the conventional representation. The digit is calculated in a redundant format with two extra leading bits (for combined unit, one extra bit is sufficient). The extra bits are used to judge if the digit produced by last cycle needs updating, since new iteration may generate carry to the previous digit.

If the extra bits are different with the least significant bits of the previous digit, then we know the previous digit has to be updated (plus 1 or 2). The scheme is shown in Figure 9.

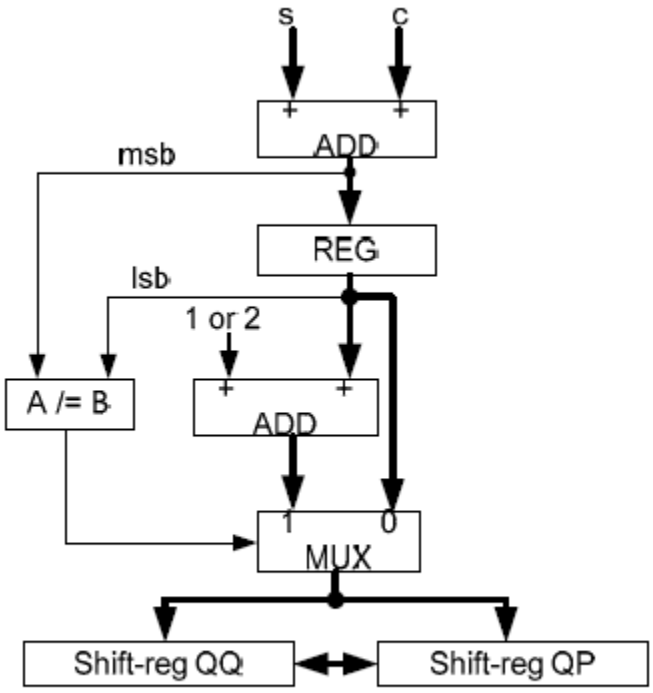

Figure 9 Scheme of on-the-fly conversion for approximation part.

The rounding of approximation part is done using the round-to-nearest scheme. However, a few cases cannot be directly rounded. A new method is presented to obtain correctly rounded result. The method consists of two steps.

1. Determine if result obtained can be directly rounded. The result could be computed with some additional accuracy so that the probability of being able to do this rounding is high.

2. In the few cases, the rounding cannot be performed, then continue with the digit-recurrence until produce the rounding bit and the corresponding final residual.

This scheme leads to the latency increases for critical cases, but these cases have very low probability. Moreover, only little additional hardware is needed since the digit-by-digit hardware is available.

3. ARCHITECTURE

Based on the algorithms of chapter 2, the design of the Architecture can have several alternatives. The design choices mainly depend on several interrelated factors, such as radix, quotient-digit set and representation of the residual. These factors affect the execution time and the complexity of implementation.

High radix reduces the number of iterations. The number of iterations of the algorithm is reduced by a factor k when going from a radix r to a radix rk. However, increase in radix produces a more complex implementation because of the quotient-digit selection and the generation of the divisor multiples.

The value of the redundancy factor (ρ) for quotient-digit set effects the complexity of the quotient- digit selection function and of the generation of the divisor multiples in an Opposite manner: a higher ρ reduces complexity of the selection function but increases complexity of the generation of the divisor multiples .

For the representation of the residual, carry-save form has the big advantage that the addition is faster than the conventional two's complement representation. Its disadvantages are that it complicates somewhat the quotient-digit selection and that it increases the number of register bits required to store the residual.

This project uses radix-4 implementation, carry-save representation for the residual and the digit-set {-2, -1, 0, 1, 2}. This choice can achieve low latency with moderate hardware overhead.

3.1 Reciprocal Architecture Design:

The digit-by-digit recurrence expression define as

$$w[j+1] = rw[j] - q_{j+1}d$$
$$q_{j+2} = SEL(rw[j+1], d)$$

The recurrences consist five steps:

1. One digit arithmetic left shift of w[j] to produce r w[j].
2. Determination of the quotient digit qj+1 by the quotient-digit selection.
3.Generation of the divisor multiple d•qj+1.
4. Subtraction of d•qj+1 from r w[j].

5. Update of the quotient q[j] to q[j+1] by the on-the-fly conversion.

These steps result in an architecture shown in the left side of Figure 10. A multiplexer is used to select initial value w[0](w[0] = 1/4) and the shifted residual r w[j]. The quotient-digit selection function requires an estimation of r w [j+1]and d (described later). This estimation needs seven most significant bits of r w[j] and three bits of d. Since r w[j] is in carry-save form, a short adder is needed to convert to conventional representation. Because digit q takes values from the digit-set {-2, -1, 0, 1, 2}, generating of the divisor multiple d•qj+1 becomes easy. The digit multiplier can be implemented as a 4-1 multiplexers. It makes the multiplication fast and simple. A fast 3-2 carry- save adder is a natural choice for residual recurrence.

According to the expression (18), the Newton-Raphson approximation recurrence is

$$E[j+1] = r^2 E[j] + q_{j+1}(2rw[j] - q_{j+1}d)$$

The recurrences consist following steps:

1. Shift 1 bit left of r w[j]to produce 2rw[j].
2. Subtraction of d•qj+1 from 2rw[j].
3. Multiplication of digit qj+1 with the result of step 2.
4. Shift 4 bits left (since r = 4) of E[j] to generate $r^2 E[j]$
5. Addition the results of step 3 and step 4 to generate E [j+1].
6. On-the-fly conversion of E [j+1] to the conventional two's complement representation.

The right side of Figure 10 shows the architecture of Newton-Raphson approximation Recurrence. Again, using a 3-2 carry-save adder for fast addition of 2rw[j] and -d•qj+1. The Addition result multiplies digit qj+1 using a 4-1 multiplexer as well. But notice that the Multiplication here is different with the multiplication in digit-by-digit part. Since the Multiplication in digit-by-digit part generates negative d•qj+1, and it is positive in approximation Part. Thus the input qj+1 to the multiplication in approximation part should be inversed. Also Notice that the bit width in approximation part is changed. Value of -d•qj+1 and 2rw[j] from digit- By-digit parts, being the two inputs to the approximation part are 56 bits. While in approximation Part, least 57 bits of E[j] needed for the iterations. It is necessary to do sign extension before the addition of recurrence. Naturally, the addition is performed by a 4-2 carry-save adder [7]. This architecture requires g+1 cycles for reciprocal computation. The first cycle performs the initialization

and computes q1. Cycle 2 to g corresponds to normal iterations and cycle g+1 is to finish the conversion and rounding.

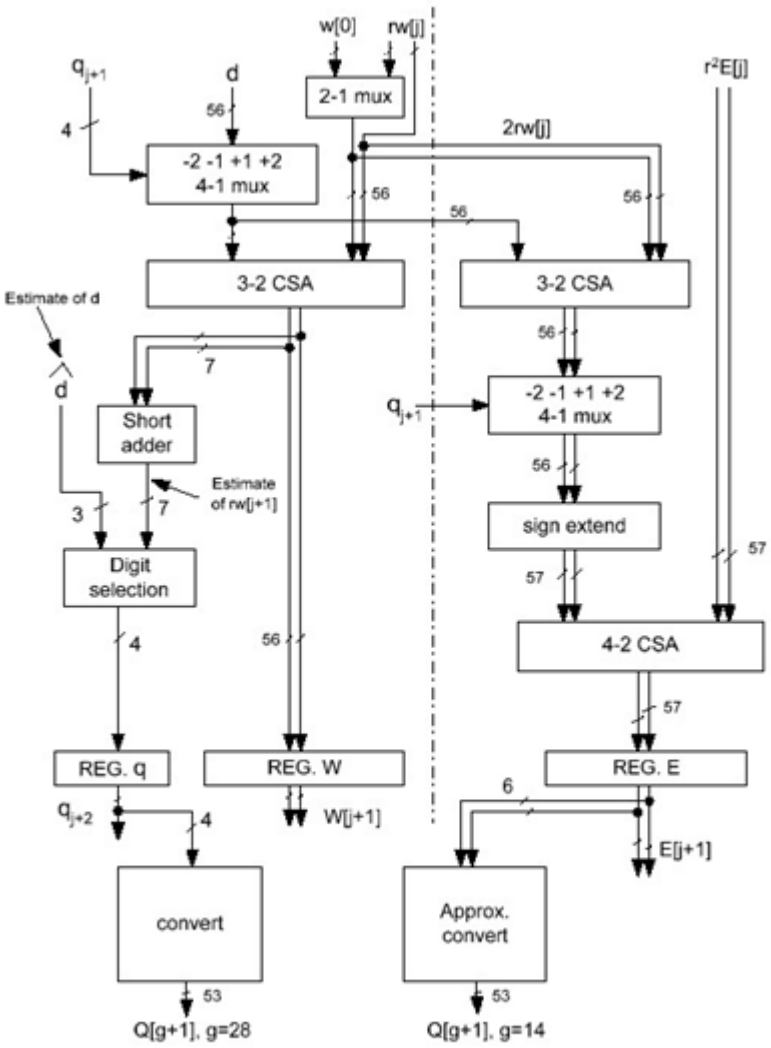

Figure 10 Architecture for reciprocal computation

3.2 Square Root Reciprocal Architecture Design:

As shown by expression (9), (10), (11), (12), the digit-by-digit recurrences of square root reciprocal consist all five steps of reciprocal recurrences and plus following steps:

1. Update C[j] to C[j+1] by shift right 2 bits.
2. Shift C[j] left 1 bit to produce 2C[j] and multiply with digit pj+1.
3. Add the result of step 2 with D[j] to perform the recurrence of D[j].
4. Multiplication to produce p2j+1C[j] used for w[j] recurrence.

The architecture for the square root reciprocal is shown in Figure 11 (This architecture is also for the combined unit). Similar to reciprocal recurrence, Several 2-1 multiplexers are used for selection between the initial value and the updated value of w [0], C [0], D [0]. Digit multipliers are implemented as 4-1 multiplexers as before. Being different with reciprocal recurrence, there are multiplications by p2j+1. This multiplication can be simplified to the selection among the multiples 0, 1, 4, and implement as 4-1 multiplexers.

In this design, D[j] and C[j] are represented in conventional two's complement form, but r w[j] is represented in carry-save form. Therefore, a 4-2 carry-save adder is used for residual recurrence, and a fast carry-propagate adder is used to update the value of D[j].

The quotient-digit selection function requires an estimation of r w[j+1]and D[j+1]. Same as reciprocal recurrence, the estimation of r w[j+1] need seven most significant bits, but convert from carry-save form to conventional representation performed by a short adder. The estimate of D[j] is obtained by the addition of the most significant bits of 2pj+1C[j] and D[j].

For the Newton-Raphson approximation part, even though the recurrence expression (20) is similar to the reciprocal, the implementation is not straightforward. Because we want to design the architecture to allow the combined implementation of reciprocal and square root reciprocal.

1. Shift r w[j] left 1 bit to produce 2rw[j] and multiply with digit pj+1, the multiplication is performed by a 4-1 multiplexer as before.
2. Shift 4 bits left (since r = 4) of H[j] to generate r2H[j].
3. Add the results of step 1 and step 2, which is performed by a 4-2 carry-save adder.
4. Multiplication to produce –p2j+1D[j]/2. The multiplication by p2j+1 has the same structure as the 4-1 multiplexer (select multiples 0, 1, 4) used in the digit-by-digit recurrence part.

5. Add the result of step 4 and step 5, which is performed by a 3-2 carry-save adder.
6. . W[j] multiplies with digit pj+1, the multiplication is performed by a 4-1 multiplexer.
7. Add the result of step 5 and step 6, which is performed by a 4-2 carry-save adder.

The sign of the digit pj+1 and the change of bit width should be taken attention as well. The result of the approximation H is converted on-the-fly. The architecture requires g+1 cycles. The first cycle performs the initialization and computes p1. Cycle 2 to g

corresponds to normal iterations and cycle g+1 is to finish the conversion and rounding. Digit-by-digit recurrence and approximation recurrence execute in overlapped manner. This results in 15 cycles for double precision float point computation. In contrast, a conventional digit-by-digit algorithm requires 28 cycles for double precision.

3.3 Combined Unit Architecture:

The combined unit architecture is based on the square root reciprocal design. Since the square root reciprocal recurrence is different with reciprocal recurrence. To make it suitable for the reciprocal computation, following steps have to do:

1. Initialize C[0] = 0 for reciprocal computation . As a result, D[j] = d.

2. Reciprocal and square root reciprocal use a single selection function. This will be discussed in next section.

3. Reciprocal approximation recurrence E does not add the term w [j+1]. Therefore, an AND gate is used to make pj+1 = 0 before the multiplexer that performs pj+1w[j], then the output of multiplexer will be zero when the reciprocal operation is performed.
4. Recurrence H requires the term –p2j+1D[j] / 2, while the reciprocal recurrence E needs –p2j+1D[j], therefore, a 2-1 multiplexer is used to select between D[j]/2 and D[j] before multiplication. A control signal OP is used to decide if the operation is reciprocal or square root reciprocal.

Figure 11 shows the full architecture of combined unit.

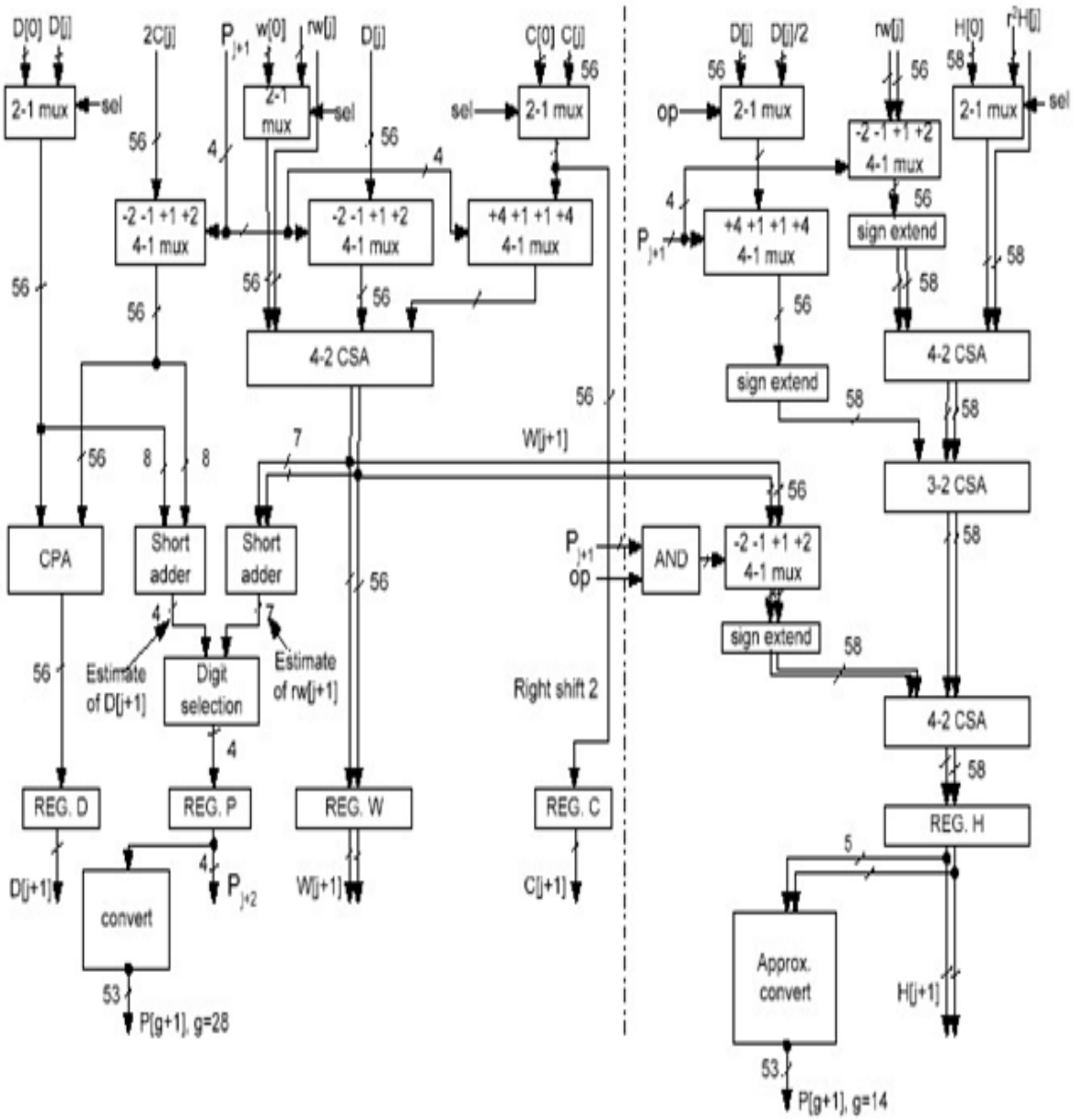

Figure 11. Architecture for reciprocal and square root reciprocal combined unit

3.4 Initialization:

This section introduced the first step of implementation, initialization for both reciprocal unit and combined unit.

Initialization for reciprocal:

There are several ways for the initialization. To assure convergence, we initialize:

$$ws[0] = 1/4$$
$$wc[0] = 0$$
$$Q[0] = 0$$
$$E[0] = 0$$

Initialization for combined unit :

To assure convergence and have the result within the interval [2,1), the following initializations are used:

For reciprocal computation,

$$Q[0] = 1 \text{ if } d > 0.75 \text{ else } 2$$
$$w[0] = 1 - dQ[0]$$
$$E[0] = Q[0](2 - dQ[0])$$

For square root reciprocal computation,

$$P[0] = 1 \text{ if } d > 0.5 \text{ else } 2$$
$$w[0] = (1/2)(1 - dP[0]_2)$$
$$D[0] = dP[0]$$
$$C[0] = (1/8)d$$
$$H[0] = (P[0]/2)(3 - dP[0]_2)$$

Table 2 lists the initial value for combined unit implementation. In the initialization formula, d is in decimal format. In practice, d as operand is represented in sign-and-magnitude format. Therefore, shifting is needed when initial values go to implementation.

Table 2 Initialization for combined unit implementation

Reciprocal computation		Square root reciprocal computation	
$d \geq 0.75$	Q[0] = 1 w[0] = 1 - d C[0] = 0 D[0] = d E[0] = 2 - d	$d \geq 0.5$	P[0] = 1 w[0] = (1/2)(1 - d) D[0] = d C[0] = (1/8)d H[0] = (1/2)(3 - d)
$d < 0.75$	Q[0] = 2 w[0] = 1 - 2d C[0] = 0 D[0] = d E[0] = 2(2 - 2d)	$d < 0.5$	P[0] = 2 w[0] = (1/2)(1 - 4d) D[0] = 2d C[0] = (1/8)d H[0] = 3 - 4d

3.5 Selection Function:

This section introduced the quotient digit selection function for radix-4 implementation. The residual is in carry-save form, represented by the sum ws and stored-carry wc. The quotient digit set is {-2, -1,0, 1, 2 }.

Selection function for reciprocal :

The quotient-digit selection depends on the truncated carry-save shifted residual $\hat{y}$ and the truncated divisor $\hat{d}$ in terms of selection constants $m_k(i)$ so that

$$q_{j+1} = k \text{ if } m_k(i) \leq \hat{y} < m_{k+1}(i)$$

where

- $i = 16\hat{d}$: $\hat{d}$ is the divisor truncated to the fourth fractional bit
- $\hat{y}$ is 4w[j] in carry-save form and truncated to the fourth fractional bit.

The selection constants are given by the following table:

Table 3 Selection constants for reciprocal

i	8	9	10	11	12	13	14	15
$m_{-2}(i)$	12	14	15	16	18	20	20	24
$m_1(i)$	4	4	4	4	6	6	8	8
$m_0(i)$	-4	-6	-6	-6	-8	-8	-8	-8
$m_{-1}(i)$	-13	-15	-16	-18	-20	-20	-22	-24

* real value = shown value/16 (fractional numbers)

Figure 12 shows an example of how the digit q_{j+1} is selected for division computation, naturlly, the method applies to the reciprocal as well. The calculation is based on the recurrence expression (3). We suppose divisor:

$$d = (0.11000101)_2$$

then

$$i = 16(0.1100)_2 = 12$$

```
4ws[0] =   000.10101111
4wc[0] =   000.00000001        ŷ[0] = 10/16     q1 = 1
  -q1d =    11.00111010
-------------------------------------------------------------------------
 ws[1] =     1.10010100
 wc[1] =     0.01010110
-------------------------------------------------------------------------
4ws[1] =   110.01010000
4wc[1] =   001.01011000        ŷ[1] = -6/16     q2 = 0
  -q2d =    00.00000000
-------------------------------------------------------------------------
 ws[2] =     1.00001000
 wc[2] =     0.10100000
-------------------------------------------------------------------------
4ws[2] =   100.00100000
4wc[2] =   010.10000000        ŷ[2] = -22/16    q3 = -2
  -q3d =    01.10001010
-------------------------------------------------------------------------
 w[3] =      0.00101010
```

Figure 12 Example of quotient digit selection.

Selection function for combined unit :

The single selection function for both reciprocal and square root reciprocal was developed in [3]. The quotient-digit selection depends on the truncated carry-save shifted residual $\widehat{rw[j]}$ and the truncated D[j], that is

$$p_{j+1} = SEL(\ \widehat{rw[j]}, \widehat{D[j]}\ )$$

where

- $\widehat{D[j]}$ is D[j] truncated to the fifth fractional bits.
- $\widehat{rw[j]}$ is the seven most significant bits of 4w[j].

The selection constants are given by table 4:

Table 4 Selection constants for reciprocal and square root reciprocal

$D_L(i)$*	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
m_{-1}	-13	-14	-14	-15	-16	-17	-17	-18	-18	-19	-21	-21	-21	-23	-24	-24
m_0	-5	-5	-5	-6	-6	-6	-7	-7	-7	-8	-8	-8	-9	-9	-9	-10
m_1	3	4	4	4	4	4	4	5	7	7	7	7	8	8	8	8
m_2	12	13	14	14	15	15	16	17	18	18	19	19	22	22	22	22

*If D_L < 16/32(> 31/32) saturate to 16/32 (31/32).
*$D_L(i)$ scaled by 32 and m_k scaled by 16.

The selection is performed as follows:

$$p_{j+1} = -2 \quad \text{if} \quad \widehat{4w} < m_{-1}(i)$$

$$p_{j+1} = k\ (-1 \le k \le 1) \quad \text{if} \quad m_k(i) \le \widehat{4w} \le m_{k+1}(i)$$

$$p_{j+1} = 2 \quad \text{if} \quad \widehat{4w} \ge m_2(i)$$

Figure 13 shows an example of how the digit p_{j+1} is selected for square root reciprocal computation. The calculation is based on the recurrence expression (9), (10) and (11). We suppose:

$$d = (0.0101001010000000)_2$$

d < 0.5, then P[0] = 2. As indicated by Table 2 and Table 4, the initialization and digit selection are shown below:

Initaialzation and computation of p_1

ws[0] = 1.1101101100000000 D[0] = 0.1010010100000000 $\widehat{D[j]}$ = 20/32, $\widehat{4w}$ = -10/16

wc[0] = 0.0000000000000000 C[0] = 0.0000101001010000 $p_1 = -1$, $P = (1.3)_4$

Iteration j = 0

ws[1] = 0.0000011010101111 D[1] = 0.1001000001100000 $\widehat{D[j]}$ = 18/32, $\widehat{4w}$ = 1/16,

wc[1] = 0.0000000000000001 C[1] = 0.0000001010010100 $p_2 = 0$, $P = (1.30)_4$

Iteration j = 1

ws[2] = 0.0001101010110000 D[2] = 0.1001000001100000 $\widehat{D[j]}$ = 18/32, $\widehat{4w}$ = 6/16,

wc[2] = 0.0000000000010000 C[2] = 0.0000000010100101 $p_3 = 1$, $P = (1.301)_4$

Iteration j = 2

ws[3] = 1.1101100111000100 D[3] = 0.1001000110101010 $\widehat{D[j]}$= 18/32, $\widehat{4w}$ = -11/16,

wc[3] = 0.0000000000110111 C[3] = 0.0000000000101001 $p_4 = -1$

Figure 13 Example of square root reciprocal computation

4. IMPLEMENTATION AND RESULTS

This chapter introduces the implementation and results are evaluated and compared with other implementation schemes. The designs of reciprocal and combined unit are described in VHDL.VHDL descriptions are simulated and synthesized using XILINX software. The critical paths are estimated and several schemes are compared with proposed scheme.

Table 5 Critical path of reciprocal unit.

QLATCH	4-1 multiplexer	CSA32	Selection	Critical path (ns)
0.46	0.19	0.17	0.74	1.56

Table 6 Critical path of combined unit

Control	2-1 multiplexer	4-1 multiplexer	CSA42	Short adder	Selection	QLATCH	Critical path (ns)
0.40	0.14	0.09	0.36	0.47	0.40	0.11	1.97

Table 5 and table 6 show that for both reciprocal and square root reciprocal operation, the critical path corresponds to the digit by digit part. this means proposed new schemes have the same cycle time as the corresponding digit by digit schemes. but the no. of iterations is only half of the digit by digit schemes alone. the total delay is reduced roughly by half.

Since the approximation part has roughly the same complexity as the conventional digit by digit part, the area of proposed schemes increase the hardware overhead by nearly 2 with respect to corresponding digit by digit schemes.

For reciprocal, the area – time ratios of higher radix schemes with respect to radix-4 scheme are described in [2]. Therefore we can compare the design in proposed scheme with more instances in an area – time space. Fig 14 shows the area – time ratios for digit by digit radix-4, radix -16 using overlapped radix and very high radix -512 schemes in comparison with the proposed design.

For the square-root reciprocal there are several implementations using radix-4 and very high radix described in [3] and [5]. A radix – 16 implementation with overlapped radix-4 iterations would be significantly more complex, because of the many conditional forms required, so we do not consider it.the scheme and implementation proposed by [3] has good area-delay figures, so take [3] as the radix-4 design reference.for the very high radix implementation, assume a cost in area of 1.5 times (a conservative figure) the cost of a very high radix reciprocal unit with the same cycle time. Fig 14 also shows the ratios corresponding to the squareroot reciprocal, using the area and delay of the conventional radix-4 reciprocal unit as reference. From the estimations it can be concluded that the proposed designs introduce aattractive points in the area - time space.

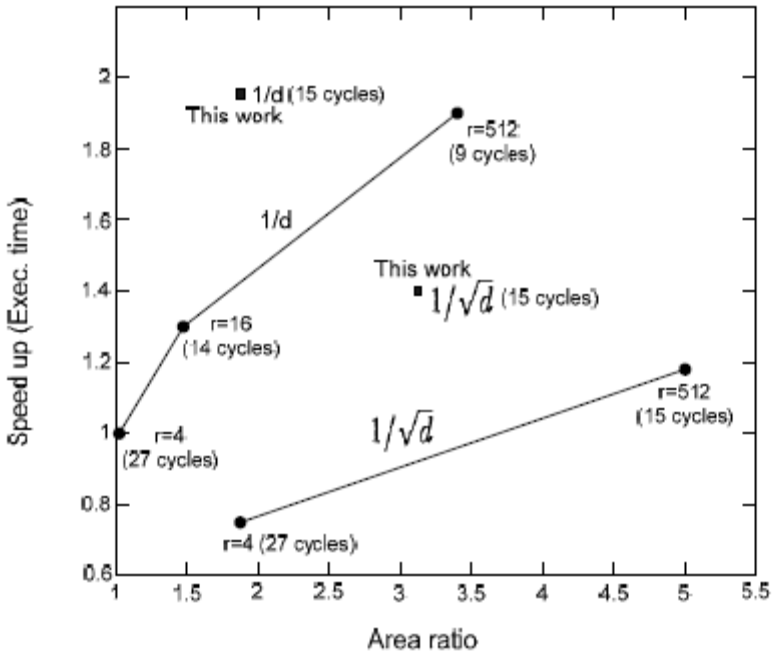

FIG.14. AREA- TIME SPACE

5. conclusion

In this project, the process of designing a combined unit for reciprocal and square root reciprocal are presented. A new algorithm is used, which combined digit by digit algorithm and newton-raphson approximation. The approximation part is performed by a digit recurrence using the digit produced by the digit by digit part. in this way two parts can execute in an overlapped manner. Consequently only half of the iterations are needed as compared to the conventional digit recurrence algorithm.

For the design of reciprocal unit and combined unit, the radix-4 implementation is used and the residual are represented in carry-save form. This choice can achieve low latency with moderate hardware overhead. To make the architecture suitable for both reciprocal and square root reciprocal computation, a signal quotient digit selection function is used which is developed in [3].

Since the proposed design produces four bits per cycle, for reciprocal, it is about 50% faster than a radix-16 implementation with overlapped radix-4 stages with an increase of 20% in area. The evaluation shows that the propose design shows the good figure in area-time space.

6. APPENDIX

A CD is attached to the end of theses report that contains VHDL code for both RECIPROCAL AND SQUARE-ROOT RECIPROCAL operations.

7.BIBLIOGRAPHY

1. E. ANTELO .P. MOTUSCHI. T. LANG. A .NANNARELLI, “Low latency digit recurrence Reciprocal and Square-root Reciprocal Algorithm and architecture”, in proceedings – 17th IEEE symposium on computer arithmetic, ARITH-17 2005 pp.147-154.
2. M.Ercegovac and T.LANG. “DIGITAL ARITHMETIC”, Morgan Kauffmann publishers, 2003.
3. T.Lang and E.Antelo,”Radix-4 Reciprocal square-root and its combination with Division and square-root”, IEEE Trans on comput.. vol 52, no.9, sept. 2003,pp.1100-1114
4. D.A.Patterson and J.L.Hennessy, “Computer organization and design-the hardware/software interface”, Morgan Kaufmann Publishers Inc., 2nd edition, 1998.
5. E.Antelo, T.Lang and J.D.Bruguera.” Computation of $\sqrt{(x/d)}$ in a very High-Radix combined division/square-root unit with scaling and selection by Rounding”, IEEE Trans on computers, vol.47, no.2, Feb. 1998, pp 152-161.
6. N.Takagi. “A hardware algorithm for computing reciprocal square root”, in proc. 15th IEEE symposium on computer arithmetic, pp 94-100, 2001.
7. WWW.XILINX.COM