The Dissertation Committee for Inwook Kong
certifies that this is the approved version of the following dissertation:

Improved Algorithms and Hardware Designs for Division by Convergence

Committee:

Earl E. Swartzlander, Jr., Supervisor

Anthony P. Ambler

Mircea D. Driga

Mohamed G. Gouda

Nur A. Touba

Improved Algorithms and Hardware Designs for Division by Convergence

by

Inwook Kong, B.S.E.E., M.E.E.E.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2009

Dedicated to my wife, Youngsook Park,

for her love and support.

Acknowledgments

I wish to thank all the people who have helped me accomplish this research. This research would not be possible without their help and support.

First of all, I would like to thank my supervisor, Dr. Earl E. Swartzlander, Jr. for his invaluable guidance, support, and expertise in this field. He has guided me throughout my research with remarkable insight and profound knowledge. Without his guidance and support, this research could not have been achieved.

I am also grateful to the members of my dissertation committee, Dr. Anthony P. Ambler, Dr. Mircea D. Driga, Dr. Mohamed G. Gouda, and Dr. Nur A. Touba. Their helpful advices on my research and their knowledge have guided me in my quest to do quality work.

I would like to thank my friends in Austin and the colleagues in the application specific processor group for their help and cheering up. I am also indebted to many people in Samsung for their support and concern for me.

Finally, I would like to thank my wife Youngsook Park, who always supports and encourages me with great love, my daughter Junghyun, who always cheers me up, and my parents and parents-in-law for their love throughout my life.

Improved Algorithms and Hardware Designs for Division by Convergence

Publication No. _____

Inwook Kong, Ph.D.
The University of Texas at Austin, 2009

Supervisor: Earl E. Swartzlander, Jr.

This dissertation focuses on improving the division-by-convergence algorithm. While the division by convergence algorithm has many advantages, it has some drawbacks, such as a need for extra bits in the multiplier and a large ROM table for the initial approximation. To mitigate these problems, two new methods are proposed here. In addition, the research scope is extended to seek an efficient architecture for implementing a divider with Quantum-dot Cellular Automata (QCA), an emerging technology.

For the first proposed approach, a new rounding method to reduce the required precision of the multiplier for division by convergence is presented. It allows twice the error tolerance of conventional methods and inclusive error bounds. The proposed method further reduces the required precision of the multiplier by considering the asymmetric error bounds of Goldschmidt dividers.

The second proposed approach is a method to increase the speed of convergence for Goldschmidt division using simple logic circuits. The proposed method achieves nearly cubic convergence. It reduces the logic complexity and delay by using an approximate squarer with a simple logic implementation and a redundant binary Booth recoder.

Finally, a new architecture for division-by-convergence in QCA is proposed. State machines for QCA often have synchronization problems due to the long wire delays. To resolve this problem, a data tag method is proposed. It also increases the throughput significantly since multiple division computations can be performed in a time skewed manner using one iterative divider.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

Division is a basic operation in many scientific and engineering applications, and two categories of division algorithms have been developed by researchers. Although division is typically an infrequent operation compared to addition and multiplication, it has been shown that if division is ignored, many applications will experience performance degradation [1]. Division algorithms can be categorized into two kinds: digit recurrent division and division by convergence. While these two kinds of algorithms have their own advantages [2], division by convergence has several advantages, such as quadratic convergence, a pipeline friendly algorithm, and a small size if the system includes a multiplier [3]. Goldschmidt division [2, 4] is a representative algorithm for the hardware implementations of division by convergence.

While division by convergence algorithms have some advantages, such as quadratic convergence and small size if the system includes a multiplier [3], they have several drawbacks. First, they do not provide an exact remainder that is necessary for correct rounding. In addition, they require extra bits in the multiplier for a correctly rounded quotient and the mitigation of the trun-

cation errors. Due to these extra bits, the precision of the multiplier should be larger than the required target precision. Second, the computation time of division by convergence heavily depends on the precision of the initial approximation to the reciprocal of denominator. Typically, the initial approximation is computed using a table look-up method, and the accuracy of the initial approximation determines the number of iterations to obtain a target precision. Although a high precision initial reciprocal table can reduce the number of iterations, it requires a large silicon area. If these drawbacks can be mitigated, the performance of the division by convergence algorithms will be improved.

In addition to improving algorithms for division by convergence in CMOS technology, a new architecture for a promising emerging nanotechnology also needs to be explored. The continued scale down of transistors confronts many challenges, such as leakage current caused by quantum mechanical tunneling of electrons. Quantum-dot Cellular Automata (QCA) [5, 6] is one of the promising emerging nanotechnologies, which may solve the problems due to the shrink of transistors. QCA has many advantages, such as low power consumption, high density, ultra fast computing, and an inherent pipeline structure by implicit D flip-flops. Due to the unique characteristics of the QCA technology, many arithmetic circuits, such as adders and multipliers, show interesting performance characteristics in the QCA technology [7, 8]. It means that an arithmetic algorithm that shows the best performance in the CMOS technology may not be the best in the QCA technology. Therefore, new arithmetic algorithms specialized for the QCA technology need to

be developed.

1.2 Research Direction

Although division by convergence has many advantages, it requires extra bits for a multiplier (more than the target precision) and a large silicon area for a high precision initial reciprocal approximation. In addition, a new architecture for an emerging nanotechnology, quantum-dot cellular automata, needs to be developed. To improve the division by convergence algorithm, three approaches are proposed in this dissertation.

The approximate quotients have to be computed with extra precision due to both truncation errors during iterations and IEEE-754 compliant rounding operations. The truncation error occurs on every iteration since multiplication results are rounded to the precision of the multiplier. The final approximate quotient always has a bounded small error due to these truncation errors. Another reason for the extra precision requirement is IEEE-754 compliant rounding. The conventional rounding method [3, 9–11] requires that the total error of the final approximate quotient should be less than $\frac{1}{4}ulp$. In other words, the number of extra bits in the multiplier has to be large enough so that the total error of the final approximate quotient is less than $\frac{1}{4}ulp$. In order to reduce the required extra precision for the approximate quotient, a new rounding method for division by convergence is proposed. The proposed rounding method applies special truncation methods at the final iteration step. This requires a minor modification to the rounding constants of the multiplier.

It allows twice the error tolerance of conventional methods and inclusive error bounds. The proposed method further reduces the required precision of the multiplier by considering the asymmetric error bounds of Goldschmidt dividers where the factors are computed using a one's complement operations.

Since the accuracy of the initial reciprocal approximation determines the number of iterations in division by convergence, an efficient reciprocal approximation method is crucial to reduce the computation time. Although a reciprocal ROM table with large precision reduces the number of iterations, it also requires a large silicon area. Therefore, many researchers have focused on implementations of efficient reciprocal approximation methods. In contrast, if the rate of convergence becomes higher than quadratic in division by convergence, the accuracy requirement for an initial reciprocal approximation can be mitigated. In other words, the target precision of a final approximate quotient can be computed from a less accurate reciprocal approximation if the speed of convergence is faster. In order to reduce the silicon area for the ROM table, a new method to speed the convergence rate using near cubic convergence is proposed. Although division with cubic convergence, the simplest higher order convergence, has been regarded as impractical due to its complexity, the new method reduces the logic complexity and delay by using an approximate squarer with a simple logic implementation and a redundant binary Booth recoder.

Although Quantum-dot Cellular Automata (QCA) [5, 6] has many advantages, there is a problem in implementing conventional sequential circuits

based on state machines. Wires in QCA have a long delay since they are implemented by QCA cells like those used to construct gates. Therefore, state machines for QCA often have synchronization problems due to the long delays between the state machines and the units (i.e., the computational circuits) to be controlled. Due to this difficulty in designing sequential circuits, it appears that a division circuit has not been researched yet while many adders and multipliers have been designed in QCA [7, 8, 12–15]. To resolve this problem, an architecture for division by convergence using a data tag method is proposed. In the new architecture for QCA, data tags are used instead of state machines, and they are associated with the data and local tag decoders generate control signals. Since each datum has a tag in the new architecture, it is possible to issue a new division command at any iteration stage of a previous issued operation, which increases the throughput of division significantly.

1.3 Dissertation Organization

The proposed research focuses on improving the division by convergence algorithms. The rest of this dissertation is organized as a total of five chapters. In Chapter 2, the conventional algorithm for division by convergence is explained with its detailed operation. In addition, the previous works to mitigate the drawbacks of division by convergence are summarized. As the first approach, a new rounding method to reduce the required multiplier precision for division by convergence is presented in Chapter 3. In Chapter 4, an approach to increase the speed of convergence for Goldschmidt division using

simple logic circuits is presented. In Chapter 5, an architecture for division by convergence in QCA is presented Finally, the summary of the research work in this dissertation and suggestions for the future research are provided in Chapter 6.

Chapter 2

Division by Convergence

The concept of division by convergence and its previous researches are shown in this chapter. In Section 2.1, the concept of Goldschmidt division is explained. Goldschmidt division is a representative division-by-convergence algorithm that is usually used for hardware implementations. In the other sections, previous researches to solve the problems shown in Section 1.2 are reviewed.

2.1 Goldschmidt Division

In this section, the concept of Goldschmidt division and its implementation are explained. The division-by-convergence algorithms explored in this dissertation are various forms of Goldschmidt division, sometimes called a series expansion method [10]. Goldschmidt dividers compute the approximate quotient by parallel multiplications. Division can be written as

$$Q = \frac{N}{D}$$

where Q is the quotient, N is the numerator, and D is the denominator. Before the first iteration of the Goldschmidt division, both N and D are multiplied by F_0 (the approximate reciprocal of D) to make the denominator close to

1. Since F_0 is produced by a reciprocal table with a limited precision, the resulting denominator has an error, ϵ. Therefore, the normalized quotient, Q_0, is

$$Q_0 = \frac{N \times F_0}{D \times F_0} = \frac{N_0}{D_0} = \frac{N_0}{1-\epsilon} \ .$$

At the first Goldschmidt iteration, N_0 and D_0 are multiplied by F_1. While F_0 is usually computed by a table look-up, the approximation for the reciprocal of D_0, F_1, is computed by an addition. This simple arithmetic computation for the approximate reciprocal is one of the advantages of Goldschmidt division. In this case, F_1 and Q_1 are computed as follows:

$$\begin{aligned} F_1 &= (2 - D_0) = 1 + \epsilon \\ Q_1 &= \frac{N_1}{D_1} = \frac{N_0 \times F_1}{D_0 \times F_1} \\ &= \frac{N_0(1+\epsilon)}{(1-\epsilon)(1+\epsilon)} = \frac{N_0(1+\epsilon)}{(1-\epsilon^2)} \end{aligned}$$

At the i-th iteration, F_i and Q_i are computed as follows:

$$F_i = (2 - D_{i-1}) = 1 + \epsilon^{2^{i-1}} \text{ for } i > 0 \tag{2.1}$$

$$Q_i = \frac{N_i}{D_i} = \frac{N_{i-1} \times F_i}{D_{i-1} \times F_i} = \frac{N_{i-1}(1+\epsilon^{2^{i-1}})}{(1-\epsilon^{2^i})} \tag{2.2}$$

As the iteration continues, N_i will converge toward Q with ever-greater precision. Since the error decreases by ϵ^{2^i} as shown in Equation (2.2), the convergence order of the Goldschmidt division is quadratic. A less than 8-bit reciprocal table [16] can make the initial error ϵ accurate enough for double precision floating-point division to be computed in three Goldschmidt iterations. In addition, the Goldschmidt division is attractive for hardware implementation

since the independent parallel multiplications can be implemented efficiently in a pipelined multiplier.

F_i for quadratic convergence is usually computed using simple logic without a real addition. If the two's complement operation used to compute F_i in Equation (2.1), it requires a carry propagating adder, which introduces a long delay. Therefore, F_i is usually computed by one's complement [3, 17] to reduce the delay although one's complement introduces a small error, 2^{-lsb}, as

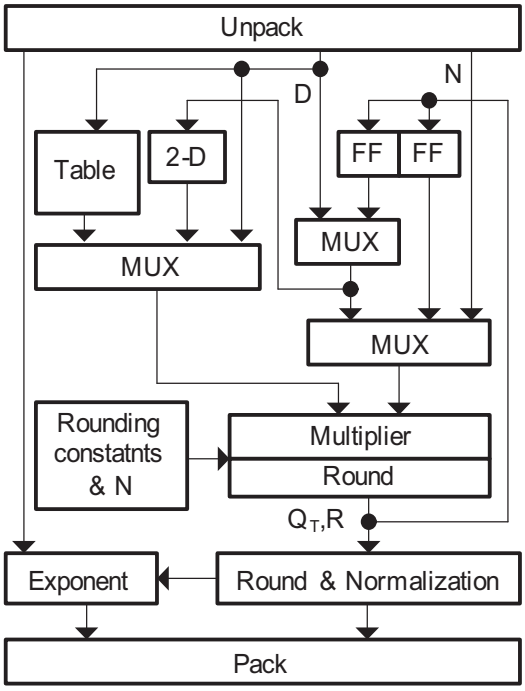


Figure 2.1: Block diagram of a typical Goldschmidt divider.

follows:

$$\begin{aligned} F_i &= 2 - D_{i-1} = \text{comp2}(D_{i-1}) \\ &= \text{comp1}(D_{i-1}) + 2^{-lsb} \\ &\approx \text{comp1}(D_{i-1}) \end{aligned} \tag{2.3}$$

Since F_i can be computed by one's complement, many Goldschmidt division applications have adopted the quadratic convergence algorithm. A typical floating point Goldschmidt divider consists of a multiplier, a reciprocal table, a computation unit for F_i, and rounding unit as shown in Figure 2.1.

2.2 Previous Works

2.2.1 Rounding Methods for Division by Convergence

Since a correction stage to compute the exact remainder is required for correct rounding in division-by-convergence, several efficient methods have been suggested for the IEEE-754 compliant rounding. The first approach is to use more than twice the precision for intermediate computation [18]. A method using a double precision accumulator was suggested to compute correctly rounded results of the Newton-Raphson iterations in the IBM RS/6000 processor [19], but the method has two drawbacks. One is that the accumulator for a multiply-and-add instruction requires twice the precision of other methods. The other is that the method needs one additional regular iteration step before the final rounding stage although the result has already reached the target precision. However, the method is very efficient for the fused multiply-

and-add architecture, and it utilizes the floating-point pipeline architecture successfully without any conditional branches for the rounding stage.

Another approach is to compute the direction of the remainder in lieu of an explicit calculation. Because this method requires only the direction of the remainder for the rounding operation, more efficient rounding circuits can be implemented. A method that computes the sign of the difference between the dividend and the product of an approximate quotient and the divisor was implemented in the TI-8847 processor [2, 20]. In this processor, quotients are computed with extra bits of precision. An enhanced method that removes the necessity of comparing the remainder in some cases using one additional correct bit [9] was developed. Furthermore, it has been shown that a reduction in the frequency of remainder comparisons is possible if several additional bits are computed [10, 21].

Although there have been improvements in the frequency of the remainder comparisons, the error tolerance of inputs to the rounding stage has not improved. Truncation of the numerator and denominator in division by convergence introduces a small amount of error at each iteration step. In the Goldschmidt division algorithm [4], this kind of error accumulates, and the total error for the approximate quotient is proportional to the number of iterations. For correct operation at the rounding stage, the absolute value of the total error has to be less than $\pm\frac{1}{4}ulp$ in current rounding methods. If this error tolerance can be increased, the intermediate precision required for iterations can be relaxed, or an iteration algorithm can be implemented with more error

margin.

2.2.2 Reduction of the Required Multiplier Precision

Goldschmidt division requires a high precision multiplier due to the extra bits required for correct rounding. To reduce the multiplier precision, several approaches have been suggested. One approach is to determine the required minimum extra bits of the multiplier by error analysis. For the AMD K-7 microprocessor, 7 extra bits were used for a 2-iteration 68-bit division after a conservative error analysis [3]. Tighter error analyses show that the required number of extra bits can be reduced further [22, 23]. Another recent approach to reduce the required multiplier precision is a method using a rectangular $m \times n$ multiplier [11], which uses a property of Goldschmidt division that the multiplicative factors are close to 1.

2.2.3 Computation Time Reduction of Division by Convergence

Many researchers have tried to reduce the computation time of division by convergence, and the most prevalent approach is to reduce the number of iterations by increasing the precision of the initial approximation to the reciprocal. It is used to compute the initial quotient value in Goldschmidt division. The number of iterations and the computation time are heavily dependent on the accuracy of the initial reciprocal approximation. Although a high precision initial reciprocal table can reduce the number of iterations, it requires a large silicon area.

Several methods have been suggested to increase the accuracy of the reciprocal approximation. The efficient implementation of the reciprocal approximation is crucial for the performance of division by convergence. Although the minimum size of the reciprocal ROM table with a certain error margin can be determined [16], a naive implementation still requires a large silicon area. One approach to reduce the area for the reciprocal table is the faithful bipartite ROM reciprocal table method [24, 25], which employs two independent parallel table look-ups and then adds the output values of the two tables to compute the final value. Since the addition can be computed at the Booth recoding part of a multiplier without a real addition, the bipartite table method has been effective and practical enough to be adopted in many division-by-convergence applications including the AMD-K7 microprocessor [3]. On the other hand, linear function approximation methods based on small piecewise-constant tables have been suggested [26, 27]. The precision of this reciprocal approximation method is high enough for a double-precision floating-point division to be computed in only one Goldschmidt iteration [28]. Although linear function approximation methods can be implemented efficiently, the computation still requires heavy use of arithmetic operations, such as multiplication, addition, and squaring.

While reciprocal approximation methods play an important role at the first stage of division by convergence, a method for removing the last iteration step has been suggested [29]. In this method, the step before the last iteration includes the amount of error reduction of the last iteration step by a table

look-up. Although this requires a table look-up and an addition, it reduces the total computation time.

While Newton-Raphson division methods typically have quadratic convergence, a cubic convergence algorithm [30] has been suggested to reduce the number of instructions. This algorithm shows that a combination of quadratic convergence and cubic convergence gives better performance than only quadratic convergence in a processor that includes a fused multiply-and-add (FMA) instruction. This algorithm is based on an assumption that the cubic convergence requires three FMA instructions and the quadratic convergence requires two instructions, which is not valid for a pipelined hardware architecture. Therefore, it is difficult to apply this algorithm to hardware implementations although the algorithm is an effective acceleration method for a system with a FMA instruction.

Chapter 3

A New Rounding Method with Improved Error Tolerance

3.1 Overview

While division-by-convergence algorithms have some advantages, such as quadratic convergence and small size if the system includes a multiplier [3], they have some drawbacks. One of the drawbacks is that they do not provide the exact remainder that is necessary for correct rounding. Therefore, a correction stage using extra bits is required for correct rounding. In order to achieve correct rounding in division-by-convergence, several efficient methods have been suggested as shown in Section 2.2.1. Another drawback is that the division by convergence algorithms require a large precision multiplier due to both the extra bits required for correct rounding and the truncation errors in the multiplier. To reduce the multiplier precision, several approaches have been suggested as shown in Section 2.2.2.

A rounding method to enlarge the allowable error tolerance is another approach to reduce the required precision of the multiplier. The truncation of the numerator and denominator in Goldschmidt division introduces a small amount of error at each iteration step. This kind of error accumulates, and

the total error of the approximate quotient is proportional to the number of iterations. For correct operation at the rounding stage with conventional rounding methods, the absolute value of the total error has to be less than $\frac{1}{4}$ of a unit in the last place ($\frac{1}{4}ulp$) in [3, 9–11]. Although a rounding method with a larger error tolerance than the conventional method has been proposed [31], it does not consider the asymmetric error bounds that arise when the factors are computed using one's complements.

The dissertation focuses on a rounding method that reduces the required precision of the multiplier. The scheme, which allows a larger error tolerance than conventional rounding methods, is implemented by applying individual special rounding constants to the final iteration stage for a rounded approximate quotient. The proposed method is further optimized based on an error analysis of the Goldschmidt divider.

The conventional rounding method is presented in Section 3.2. In Section 3.3, the new rounding method is presented. Section 3.4 shows an error analysis of a Goldschmidt divider and shows how the new rounding method can be optimized by the error analysis result. Finally, the verification methodology and the implementation are explained in Section 3.5.

3.2 Current Rounding Method

The maximum allowable error of the conventional rounding methods for the Goldschmidt division algorithm is $\pm\frac{1}{4}ulp$ [3, 9–11]. This rounding method is based on a remainder comparison using one correct additional bit, and the

error tolerance is bounded by

$$-2^{-(n+2)} < Q - Q_A < +2^{-(n+2)} \tag{3.1}$$

where Q is the infinitely precise quotient and Q_A is the approximate quotient computed by iterations. Before defining n, it is necessary to mention some assumptions and definitions used in this dissertation. First, it is assumed that the intermediate numbers are not normalized to $[1, 2)$, which does not cause loss of generality. Second, the number of bits in a bit string is defined as the number of digits below the binary point. Third, n is the number of bits in a machine representable number, so a unit in the last place (ulp) is 2^{-n}. For the double precision IEEE-754 floating-point format, n is 53 since a guard bit is included. Finally, the internal precision is k-bits. k is larger than n since the intermediate numbers require extra precision to satisfy the error tolerance bounds.

The relationships between Q_A and the range of its corresponding Q are shown in Figure 3.1. Q_A within a 1-*ulp* range is divided into four groups that can follow different rounding rules, and these rounding rules are repeated at every *ulp*. `0` and `1` on the axis in Figure 3.1 mean digits at the $(n+1)$-th bit, and X, Y, and Z are the digits at the n-th bit, *ulp*.

Three rounding modes (RNE, RI, and RZ) are implemented as shown in Table 3.1 because the four IEEE rounding modes (RNE, RPI, RMI, and RZ) can be realized using the 3 rounding modes [32]. Q_T is an $(n+1)$-bit number used for computation of the correctly rounded quotient and the remainder

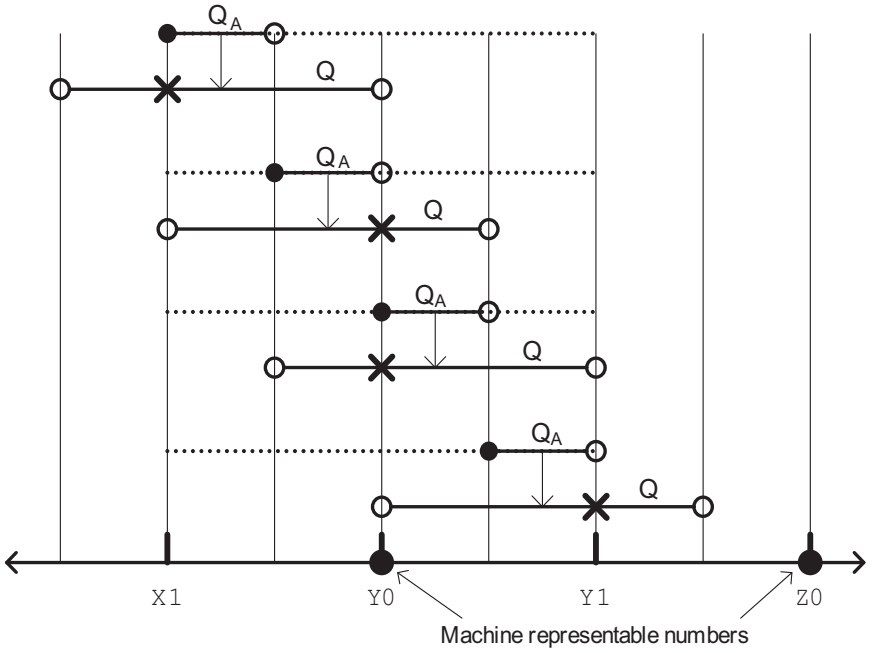


Figure 3.1: Q_A, Q, and Q_T in the conventional rounding method.

direction. The $\frac{1}{2}ulp$ bit in Table 3.1 is the $(n{+}1)$-th bit of Q_T. $Trunc$, Inc, and Dec in Table 3.1 mean the truncation of Q_T to n-bits, the increment by $1ulp$ of the truncated Q_T, and the decrement by $1ulp$ of the truncated Q_T. The Q_T corresponding to each Q_A span is indicated by an $\times$ mark in Figure 3.1, and is computed as follows:

$$\begin{aligned} Q_R &= Q_A + 2^{-(n+2)} \\ Q_T &= \text{truncation of } Q_R \text{ to } (n+1)\text{-bits} \end{aligned}$$

The relationship between Q and Q_T is

$$-2^{-(n+1)} < Q - Q_T < +2^{-(n+1)} \ .$$

Table 3.1: Conventional method rounding rule details

$\frac{1}{2}ulp$ bit	Remainder	RNE	RI	RZ
0	> 0	Trunc.	Inc.	Trunc.
0	$= 0$	Trunc.	Trunc.	Trunc.
0	< 0	Trunc.	Trunc.	Dec.
1	> 0	Inc.	Inc.	Trunc.
1	$= 0$	-	-	-
1	< 0	Trunc.	Inc. *	Trunc.

The remainder in Table 3.1 is computed using Q_T by

$$R = N - Q_T \times D$$

where R = remainder, N = numerator, and D = denominator. An exact halfway quotient, at which the $(n+1)$-th bit is 1 and the remainder is 0, is not considered in Table 3.1 because the case cannot occur if N and D are n-bits wide [9]. Since the sign of R is required for the rounding operations, only the n-th bit and the sticky bit of R are examined.

3.3 New Rounding Method

3.3.1 Inclusive Error Bounds

In the conventional rounding method introduced in Section 3.2, both endpoints of the error tolerance interval are exclusive and cannot be inclusive [9] as shown in Inequality (3.1). If both endpoints of the $(Q - Q_A)$ interval become inclusive, the lower endpoints of Q in Figure 3.1 will change from ex-

clusive to inclusive, but the upper endpoints of Q are still exclusive. Therefore, the relationship between Q_T and Q will be as follows:

$$-2^{-(n+1)} \leq Q - Q_T < +2^{-(n+1)}$$

In this slightly extended $(Q - Q_A)$ range, the case of the RI mode that is marked by * in Table 3.1 operates incorrectly. For example, if the lower endpoint of the 4-th Q ($\texttt{Y0} < Q < \texttt{Y1} + 2^{-(n+2)}$) in Figure 3.1 becomes inclusive and Q is exactly `Y0`, the correctly rounded quotient for the RI mode should be `Y0`. However, the rounded quotient computed by Table 3.1 is `Z0`, which is incorrect.

For correct rounding at the above case, a new rounding algorithm using

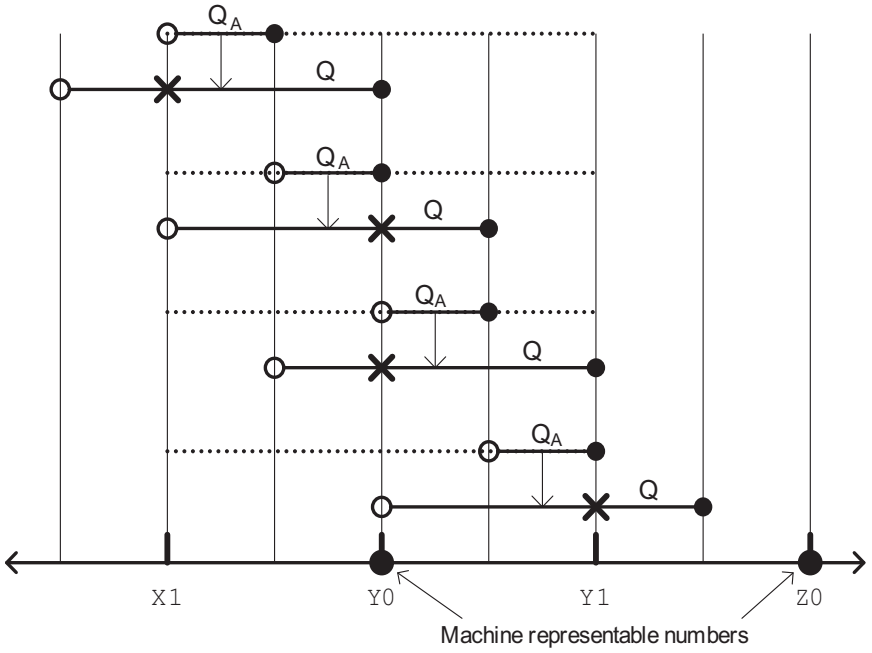


Figure 3.2: RI rounding mode with inclusive error bounds.

a different rounding constant for the RI mode is proposed. The inclusive lower endpoint of Q becomes exclusive, and the exclusive upper endpoint of Q becomes inclusive as shown in Figure 3.2. Since Q cannot be in the 4-th Q range like the problem case, the problem when $Q = \texttt{Y0}$ is solved. This new rounding algorithm is implemented by slight modification of the rounding constant for RI mode as follows:

$$\begin{aligned} Q_R^{'} &= \begin{cases} Q_A + 2^{-(n+2)} - 2^{-k} & \text{for RI} \\ Q_A + 2^{-(n+2)} & \text{for RNE \& RZ} \end{cases} \\ Q_T^{'} &= \text{truncation of } Q_R^{'} \text{ to } (n{+}1)\text{-bits} \end{aligned}$$

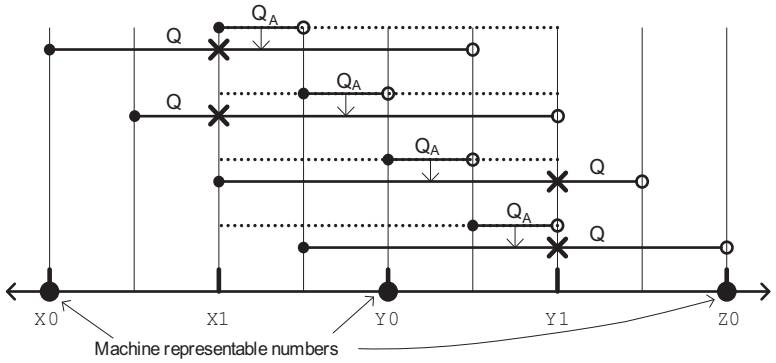
where k is the least significant bit of the internal precision. Due to extra bits to satisfy the error tolerance bounds, the internal precision, k-bits, is larger than n-bits. The remainder is computed using $Q_T^{'}$, and the correctly rounded Q is determined by the rounding rules in Table 3.1.

As a result, the error tolerance for this improved rounding mode will be

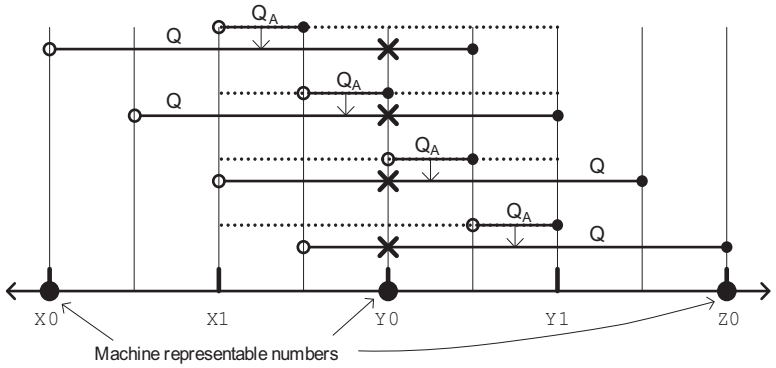
$$-2^{-(n+2)} \leq Q - Q_A \leq +2^{-(n+2)} \ .$$

3.3.2 Twice the Error Bounds with Inclusive Endpoints

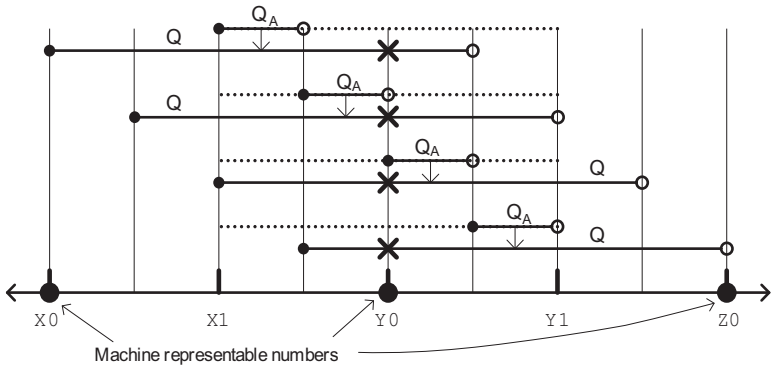
The rounding algorithm proposed in Section 3.3.1 can be expanded to accommodate a larger allowable error tolerance of $(Q - Q_A)$. The error tolerance of the conventional rounding method is limited by the RI mode in the first place. Since the first limitation on the error tolerance is removed by the algorithm in Section 3.3.1, it is possible to expand the allowable error range of $(Q - Q_A)$ beyond $\pm\frac{1}{4}ulp$.



(a)



(b)



(c)

Figure 3.3: Q, Q_A, and Q_T'' in the new rounding method. (a) RNE mode, (b) RI mode, (c) RZ mode.

Table 3.2: New rounding rules

Remainder	RNE	RI	RZ
> 0	Inc.	Inc.	Trunc.
$= 0$	Trunc.	Trunc.	Trunc.
< 0	Trunc.	Trunc.	Dec.

The new rounding algorithm for the expanded error bounds requires new rounding rules based on a new remainder computation. Q_A is converted into $Q_T^{''}$ differently according to each rounding mode as shown in Figure 3.3. The $Q_T^{''}$ corresponding to each Q_A span is indicated by an $\times$ mark in Figure 3.3. Using $Q_T^{''}$, the remainder $R^{''}$ is computed as $R^{''} = N - Q_T^{''} \times D$. The new range of $Q_T^{''}$ for each rounding mode is as follows:

$$\begin{array}{ll} -2^{-n} \leq Q - Q_T^{''} < +2^{-n} & \text{for RNE \& RZ modes} \\ -2^{-n} < Q - Q_T^{''} \leq +2^{-n} & \text{for RI mode} \end{array}$$

Finally, the correctly rounded Q is determined using the rounding rules in Table 3.2 based on $Q_T^{''}$ and the sign of the remainder $R^{''}$. The rounding rule table is simpler than the conventional method in Table 3.1 because each different truncation method to compute $Q_T^{''}$ has already been applied according to the rounding mode. As a result, the new error tolerance is

$$-2^{-(n+1)} \leq Q - Q_A \leq +2^{-(n+1)} \ .$$

3.3.3 Extension for Asymmetric Error Bounds

If the approximate quotient Q_A has asymmetric error bounds, the maximum absolute error can be reduced by shifting Q_A by adding a constant value.

In Goldschmidt dividers, asymmetric error bounds may be caused by computing the factor using a one's complement operation instead of a two's complement operation. Although the error bounds $Q - Q_A$ were used to clearly understand the rounding mode diagrams in the previous sections, the error of the approximate quotient is the inverse, specifically $E = Q_A - Q$. If the error bounds are asymmetric as

$$-B - B_{bias} \leq E \leq +B - B_{bias} \tag{3.2}$$

where $B > 0$, the maximum absolute error of Q_A is $B + |B_{bias}|$. However, if Q_A is shifted into $Q_A^{''}$ by adding B_{bias}, the effective error bounds for the new rounding algorithm will be

$$-B \leq Q_A^{''} - Q \leq +B \tag{3.3}$$

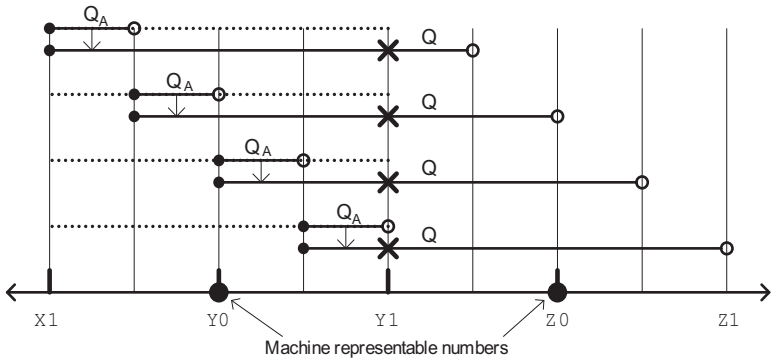
where $Q_A^{''} = Q_A + B_{bias}$.

Since only the error bounds of Q_A are shifted by B_{bias} for the rounding algorithm in order to reduce the maximum absolute error, the other procedures for rounding are the same as those of Section 3.3.2 except for applying $Q_A^{''}$ instead of Q_A.

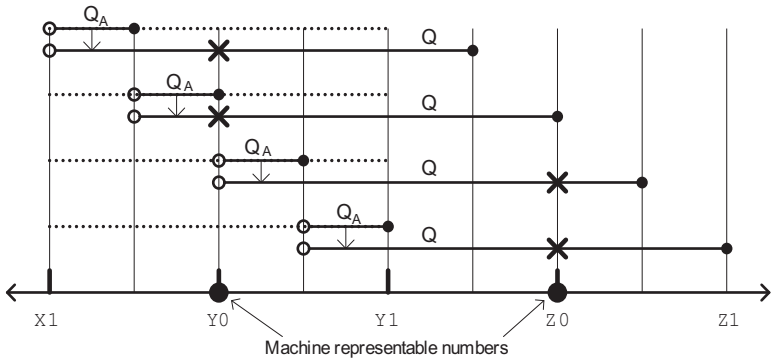
If the error of an approximate quotient is unidirectional as an instance of asymmetric error bounds, the error tolerance is

$$0 \leq Q - Q_A \leq +2^{-(n)} \ . \tag{3.4}$$

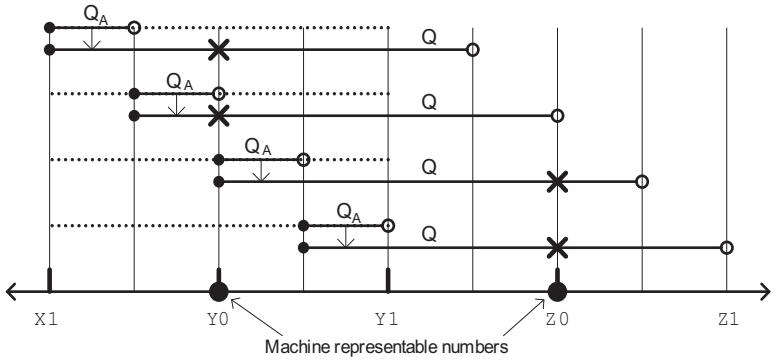
In this case, Q_A should be converted into $Q_T^{''}$ as shown in Figure 3.4 according to each rounding mode. Since the error bounds of the unidirectional errors in



(a)



(b)



(c)

Figure 3.4: Q, Q_A, and Q_T'' in the new rounding method for unidirectional errors. (a) RNE mode, (b) RI mode, (c) RZ mode.

Inequality (3.4) is

$$-2B \leq Q_A - Q \leq 0 \ ,$$

the rounding method in Figure 3.4 can be accomplished effectively by setting B_{bias} in Inequality (3.2) as B.

3.3.4 Details of the New Rounding Method

The new rounding method for the expanded error tolerance with asymmetric bounds is computed through two steps: the conversion from Q_A to $Q_T^{''}$ and the rounding rules based on both $Q_T^{''}$ and the sign of the remainder.

In the first step, Q_A that may have an error biased by B_{bias} is converted into $Q_T^{''}$ as shown in Figure 3.3. The conversion from Q_A to $Q_T^{''}$ requires a different rounding constant and truncation for each rounding mode as shown in Table 3.3. Since B_{bias} to compute $Q_A^{''}$ in Inequality (3.3) is merged into the rounding constants, there is no additional computation load for processing the

Table 3.3: Conversion from Q_A to $Q_T^{''}$

Type	Truncation method for $Q_T^{''}$
RNE	1 $Q_R = Q_A + B_{bias}$
	2 Q_R is truncated to n-bits
	3 $Q_T^{''} = Q_R \mid 2^{-(n+1)}$
RI	1 $Q_R = Q_A + (2^{-(n+1)} - 2^{-k} + B_{bias})$
	2 $Q_T^{''}$ = truncation of Q_R to n-bits
RZ	1 $Q_R = Q_A + (2^{-(n+1)} + B_{bias})$
	2 $Q_T^{''}$ = truncation of Q_R to n-bits

asymmetric error bounds. In addition, the rounding constants are likely to be $(k-n)$ bits long since B_{bias} is within an order of 2^{-k}, which will be shown in Section 3.4.2.

In the second step, $Q_T^{''}$ is converted into the correctly rounded Q using rounding rules based on the sign of the remainder $R^{''}$ in Table 3.2. The remainder in Table 3.2 is computed using $Q_T^{''}$ as

$$R^{''} = N - Q_T^{''} \times D \ .$$

Although $Q_T^{''}$ is truncated to n-bits for the RI mode and the RZ mode, $Q_T^{''}$ is an $(n+1)$-bit string, so a zero is concatenated to the end of the string in the two modes. If $Q_T^{''}$ is greater than or equal to 1.0, $Q_T^{''}$ is an n-bit string for floating-point normalization.

3.4 Approximation Error Bounds of Goldschmidt Dividers

3.4.1 Goldschmidt Divider

For the verification of the proposed rounding method, the 3-iteration double precision floating-point Goldschmidt divider is implemented in this dissertation. Before the iteration steps of the Goldschmidt division, both N and D are multiplied by F_0. Since F_0 is looked up from a 7-bit ROM table, so the maximum approximation error of F_0, $max(\epsilon)$, is $2^{-7.4}$ [16, 29]. After

normalizing the denominator to 1 using the ROM table, N_0 and D_0 are

$$\begin{aligned} N_0 &= N \times F_0 \\ D_0 &= D \times F_0 = 1 - \epsilon \ . \end{aligned}$$

At the i-th step, N_i and D_i are multiplied by F_i. F_i (i>0) is computed using the one's complement method in order to reduce the delay [3, 17]. In this case, F_i, N_i, and D_i are as follows:

$$\begin{aligned} F_i &= (2 - D_{i-1}) = 1 + \epsilon^{2^{i-1}} \\ N_i &= N_{i-1} \times F_i = N_{i-1}(1 + \epsilon^{2^{i-1}}) \\ D_i &= D_{i-1} \times F_i = (1 - \epsilon^{2^i}) \end{aligned}$$

As the iteration continues, the truncation errors in the multiplier during computing N_i and D_i are accumulated. In addition to the truncation errors, the final error of the approximate quotient is affected by the error due to one's complement operations in Equation (2.3).

3.4.2 Error Bounds for the Goldschmidt Divider

The maximum error bounds of Q_A are required for determining two parameters: the number of the extra bits $(k - n)$ and the B_{bias} in Inequality (3.2). The maximum error bounds of the approximate quotients are analyzed by the maximum error scenario.

The maximum error scenario for the approximate quotients includes two cases: the maximum positive error bound (E_{max+}) and the maximum

negative error bound (E_{max-}). The errors are caused by rounding N_i and D_i to the internal precision of the multiplier, k ($k > n$). The maximum positive error occurs when N_i is maximized and D_i is minimized by rounding (round to nearest) at each iteration step. Conversely, the maximum negative error occurs when N_i is minimized and D_i is maximized.

The maximum positive error bound, E_{max+}, occurs when N_i is maximized and D_i is minimized at each iteration step. The maximum N_0 and the minimum D_0 that occurs due to truncation errors, e_{lsbn0} and e_{lsbd0}, are

$$\begin{aligned} N_{0max} &= N_0 + 0.5e_{lsbn0} \\ D_{0min} &= (1-\epsilon) - 0.5e_{lsbd0} \end{aligned}$$

where $e_{lsbn0} = e_{lsbd0} = 2^{-k}$. k is the last bit position of the internal precision including the extra bits. F_1 is computed by

$$F_1 = 2 - D_{0min} - 2^{-k} = 1 + \epsilon + \frac{1}{2}e_{lsbd0} - 2^{-k} \tag{3.5}$$

where $e_{lsbd0} = 2^{-k}$. Since F_1 is computed using a one's complement operation, the term 2^{-k} is subtracted in Equation (3.5). By using N_{0max}, D_{0min} and F_1, N_{1max} and D_{1min} are estimated as

$$\begin{aligned} N_{1max} &\cong N_0(1+\epsilon) + 0.5(1-N_0)2^{-k} + 0.5e_{lsbn1} \\ D_{1min} &\cong 1 - \epsilon^2 - 2^{-k} - 0.5e_{lsbd1} \end{aligned}$$

where $e_{lsbn1} = e_{lsbd1} = 2^{-k}$. In the same manner, N_{2max} and D_{2min} are esti-

mated as

$$\begin{aligned} N_{2max} &\cong N_0(1+\epsilon)(1+\epsilon^2) + 2^{-k} + 0.5e_{lsbn2} \\ D_{2min} &\cong 1 - \epsilon^4 - 2^{-k} - 0.5e_{lsbd2} \end{aligned}$$

where $e_{lsbn2} = e_{lsbd2} = 2^{-k}$. At the final iteration, N_{3max} and D_{3min} are estimated as

$$\begin{aligned} N_{3max} &\cong N_3 + 0.5(3+N_2)2^{-k} + 0.5e_{lsbn3} \\ D_{3min} &\cong 1 - \epsilon^8 - 2^{-k} - 0.5e_{lsbd3} \end{aligned}$$

where $e_{lsbn3} = e_{lsbd3} = 2^{-k}$. Since $max(Q_A)$ is N_{3max}, the maximum positive error bound is as follows:

$$\begin{aligned} E_{max+} &= max(Q_A) - Q \cong N_{3max} - N_3(1+\epsilon^8) \\ &= N_3 + (\frac{1}{2}N_2 + 2)2^{-k} - N_3(1+\epsilon^8) \\ &= (\frac{1}{2}N_2 + 2)2^{-k} - N_3\epsilon^8 \end{aligned}$$

$min(Q_A)$, which is required to determine E_{max-}, is also estimated as shown

Table 3.4: Maximum error bounds of the approximate quotient for the Goldschmidt divider as implemented

Bounds	Error $(E = Q_A - Q)$	$Q < 1$	$Q \geq 1$
E_{max+}	$(\frac{1}{2}N_2 + 2)2^{-k} - N_3\epsilon^8$	$2.5 \cdot 2^{-k}$	$3 \cdot 2^{-k}$
E_{max-}	$-(\frac{5}{2}N_2 + 2)2^{-k} - N_3\epsilon^8$	$-4.5 \cdot 2^{-k} - \epsilon_0^8$	$-7 \cdot 2^{-k} - 2\epsilon_0^8$

in Table 3.4 using the same procedure. The errors in Table 3.4 are simplified by assuming that ϵ_0 is $max(|\epsilon|)$ and N_2 is less than or equal to Q.

If the correctly rounded Q is greater than or equal to 1, then Q_A is normalized and the $(n-1)$-th bit is a *ulp* instead of the n-th bit. Thus, the n-th bit serves as an additional bit. This extra bit reduces the error by half. For example, although E_{max-} for $Q \geq 1$ is $-7 \cdot 2^{-k} - 2\epsilon^8$ in Table 3.4, the maximum negative error after normalization is $-\frac{7}{2} \cdot 2^{-k} - \epsilon^8$. Since ϵ^8 is less than 2^{-k} practically, the maximum error bounds for $Q < 1$ in Table 3.4 are the main concern.

If the factor F_i is computed using a two's complement operation, the maximum positive error bound and the maximum negative error bound are calculated by removing the term 2^{-k} in Equation (3.5). As a result, the two error bounds can be estimated using the same procedure as shown in Table 3.5.

Table 3.5: Maximum error bounds of the approximate quotient when F_i is computed by a two's complement operation

Bounds	Error $(E = Q_A - Q)$	$Q < 1$	$Q \geq 1$
E_{2max+}	$(\frac{3}{2}N_2 + 2)2^{-k} - N_3\epsilon^8$	$3.5 \cdot 2^{-k}$	$5 \cdot 2^{-k}$
E_{2max-}	$-(\frac{3}{2}N_2 + 2)2^{-k} - N_3\epsilon^8$	$-3.5 \cdot 2^{-k} - \epsilon_0^8$	$-5 \cdot 2^{-k} - 2\epsilon_0^8$

3.4.3 Error Bounds Comparison by the F_i Computation Methods

Before applying the new rounding method, the maximum error bounds in Section 3.4.2 are evaluated with 10^9 random double precision floating-point divisions. The approximation errors, E, are computed by modifying the remainder equation as follows:

$$E = Q_A - Q = \frac{Q_A \times D - N}{D}$$

The error histogram in Figure 3.5 shows the error distributions of four cases: C2Q0 (2's complement for F_i, $Q < 1$), C2Q1 (2's complement for F_i, $Q \geq 1$), C1Q0 (1's complement for F_i, $Q < 1$), and C1Q1 (1's complement for F_i, $Q \geq 1$).

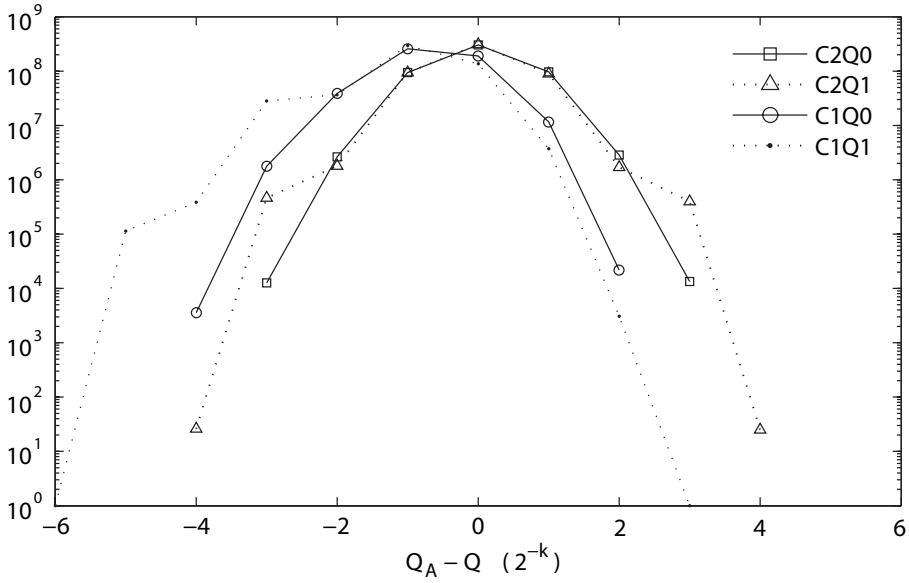


Figure 3.5: Histogram of E for 10^9 random divisions.

The results show the same trends that were predicted by the maximum error analyses in Section 3.4.2. As in the error analyses, the two's complement cases show symmetrical error distributions. On the other hand, the one's complement cases show asymmetrical error distributions. The error distribution of the one's complement case for $Q < 1$ are shifted left by 2^{-k} compared to the two's complement case.

3.4.4 Parameters for the New Rounding Method

Using the error analysis result in Section 3.4.2, the two parameters, B_{bias} and k, required to implement the new rounding method and the multiplier are determined. Since the maximum positive error bound (E_{max+}) is $2.5 \cdot 2^{-k}$ and the maximum negative error bound (E_{max-}) is $-4.5 \cdot 2^{-k} - \epsilon^8$ as shown in Table 3.4, B_{bias} in Inequality (3.2) is

$$B_{bias} = +2^{-k} + \frac{1}{2}\epsilon_0^8 \cong +2^{-k} \tag{3.6}$$

where $\epsilon_0 = max(|\epsilon|)$. Since the quantization step size for B_{bias} has to be 2^{-k} for hardware implementation, the term $\frac{1}{2}\epsilon_0^8$ in Equation (3.6) is removed. In addition, the maximum error bound B in Inequality (3.3) becomes $3.5 \cdot 2^{-k} + \epsilon_0^8$.

The number of extra bits for the multiplier is determined to insure that the maximum absolute error of $Q_A^{''}$ is less than or equal to $\frac{1}{2}ulp$. Since Q_A is shifted by B_{bias} to $Q_A^{''}$, the maximum absolute error of $Q_A^{''}$ is

$$|E_{max}^{''}| = max(|Q_A^{''} - Q|) = B$$

where $Q_A^{''} = Q_A + B_{bias}$. Since the new rounding method requires that $|E_{max}^{''}|$ is less than or equal to $\frac{1}{2}ulp$, the number of the extra bits for correct rounding should satisfy the following conditions:

$$\begin{aligned} |E_{max}^{''}| &= (3.5) \cdot 2^{-k} + \epsilon_0^8 = 2^{-k+1.81} + \epsilon_0^8 \\ |E_{max}^{''}| &\leq \frac{1}{2} \cdot 2^{-n} = 2^{-n-1} \end{aligned}$$

Since ϵ_0^8 for the 7-bit ROM table is $2^{-59.2}$ [16, 29] and n is 53, the number of the extra bits for 3-iteration double precision Goldschmidt divider is 3 bits ($k \geq n + 3$).

As a result, the multiplier precision for the 3-iteration double precision floating-point Goldschmidt divider should be at least 57 bits ($k = 56$), which consists of a 53-bit significand, a 1-bit guard, and 3 extra bits. In this case, the maximum absolute error of $Q_A^{''}$ for $Q < 1$ is as follows:

$$|E_{max}^{''}| = max(|Q_A^{''} - Q|) = (3.61) \cdot 2^{-k} \tag{3.7}$$

3.4.5 Comparison with Other Methods

The proposed rounding method implements 3-iteration double precision floating-point Goldschmidt divider with 3 extra bits. The number of the extra bits of other rounding methods for the same condition is either 5 or 4 bits as shown in Table 3.6.

Table 3.6: Required extra bits for each rounding method

	Current method [3, 9–11, 22]	K. & S. [31]	Proposed method
Extra bits	5 bits	4 bits	3 bits

3.5 Verification and Simulation Results

3.5.1 Implementation and Verification

The Goldschmidt divider with the proposed rounding method has been implemented and verified using SystemC 2.2.0 as shown in Figure 3.6. The new rounding method is implemented by applying individual special truncation methods to the final iteration stage in order to compute the rounded

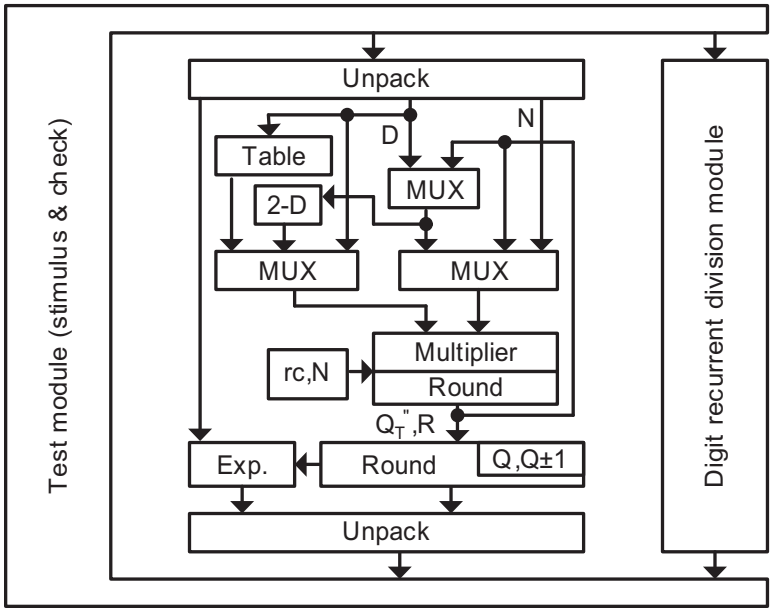


Figure 3.6: Goldschmidt divider and verification environment.

approximate quotient (Q_T''). It requires a minor modification to a conventional Goldschmidt divider because it injects proper rounding constants into the carry save adder (CSA) tree of the multiplier. Since 3 extra bits are required for IEEE-754 compliant rounding as shown in Section 3.4.4, the multiplier precision for the double precision floating-point Goldschmidt divider should be at least 57 bits, which consists of a 53-bit significand, a 1-bit guard, and 3 extra bits. Since B_{bias} is 2^{-k}, the rounding constants in Table 3.3 become simpler than those used in [31].

The verification of the new rounding method consists of two parts. First, the final results of the four IEEE rounding modes were checked by performing 10^{10} divisions with random double precision floating-point inputs. Second, the final results were also checked by exhaustive 17-bit precision fixed-point numbers (a total of 2^{32} test vectors) to insure that no special cases were missed by the random test vectors. Since exhaustive verification using double precision floating-point vectors is not feasible, the exhaustive 17-bit precision test is a practical alternative. To support this verification, the Goldschmidt divider model is designed to support variable precision. In addition, a digit recurrent divider model computes reference quotients in parallel because the X87 FPU does not support 17-bit precision division. During the two verification steps, all the errors are less than the maximum absolute error derived in Section 3.4.4.

3.5.2 Simulation Results

The validity of the maximum error bounds is checked via a simulation for both the two verifications steps in Section 3.5.1. The approximation error E'' is computed by modifying the remainder equation as follows:

$$E'' = Q''_A - Q = \frac{Q''_A \times D - N}{D}$$

E'' should be less than or equal to $4 \cdot 2^{-k}$ ($\frac{1}{2}ulp$) for the correct rounding operation of the proposed method.

The error histogram using the 10^{10} random double precision floating-point test vectors in Figure 3.7 shows that B_{bias} is effective to reduce the

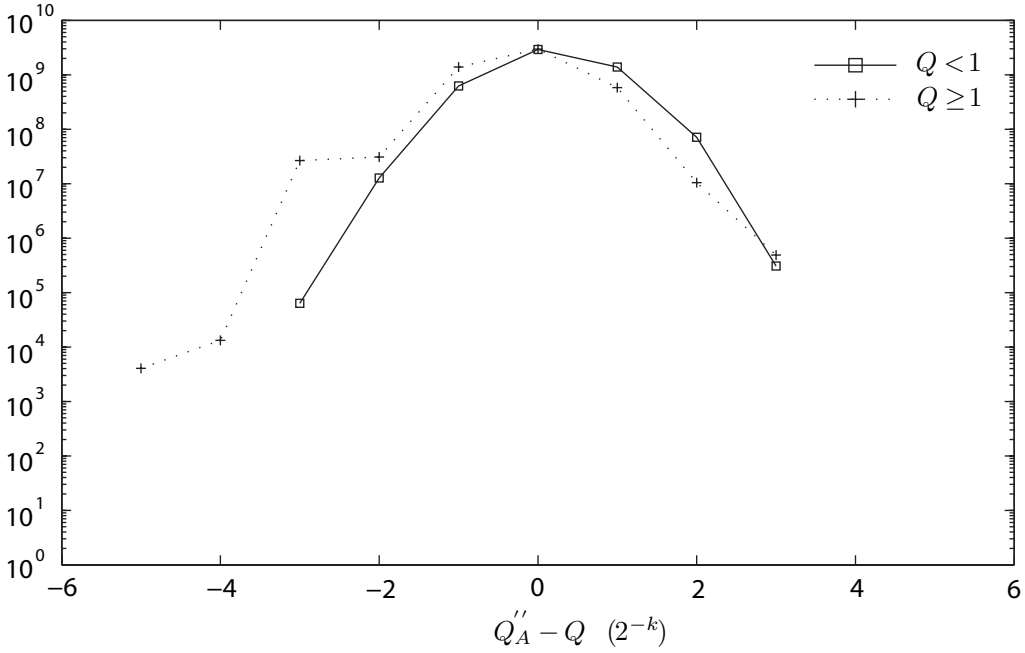


Figure 3.7: Histogram of E'' by 10^{10} random input vectors.

Table 3.7: Maximum error bounds of E'' during the simulation

Bounds	10^{10} random numbers		Exhaustive test vectors	
	$Q < 1$	$Q \geq 1$	$Q < 1$	$Q \geq 1$
E''_{max+}	$3 \cdot 2^{-k}$	$3 \cdot 2^{-k}$ *	$3 \cdot 2^{-k}$	$3 \cdot 2^{-k}$ *
E''_{max-}	$-3 \cdot 2^{-k}$	$-5 \cdot 2^{-k}$ *	$-2 \cdot 2^{-k}$	$-3 \cdot 2^{-k}$ *

*The effective errors are halved after floating-point normalization.

maximum absolute error of Q_A. Since the quantization step size of E'' is 2^{-k}, the error histogram shows that all the errors of Q''_A are bounded inside $\pm 4 \cdot 2^{-k}$ ($\frac{1}{2}ulp$) as expected.

The maximum approximation errors have been checked during all the verification steps as shown in Table 3.7. All the approximation errors of Q''_A are bounded by the maximum absolute error in Equation (3.7).

3.6 Summary

A new rounding method for division by convergence, which allows a larger error tolerance compared to the conventional rounding method, has been presented. Due to the improved error tolerance, iteration algorithms may have more error margin with the same internal precision. In this dissertation, the proposed rounding method reduces the required multiplier precision effectively. In addition to the large error tolerance of the proposed rounding method, it fully utilizes the characteristics of the approximation error bounds

to reduce the required precision of the multiplier. It can be implemented by minor modifications of the rounding constants in the multiplier. It enables a 3-iteration double precision floating-point Goldschmidt divider to be implemented using only 3 extra bits even though the factors during the iterations are computed using one's complement operation. It has been verified using a SystemC model of the Goldschmidt divider. The maximum error span was checked both by analysis and via simulation. Verifications were performed using both 10^{10} random double precision floating-point test vectors and an exhaustive suite of 17-bit precision test vectors.

Chapter 4

Division with Faster than Quadratic Convergence

4.1 Overview

The most prevalent approach to reduce the computation time of division by convergence has been to reduce the number of iterations by increasing the precision of the initial approximation to the reciprocal. Since an accurate initial approximation requires a large silicon area, many approaches for the efficient implementation of the initial approximation have been suggested as shown in Section 2.2.3. In addition to reducing the silicon area for the initial approximation, another approach to remove the last iteration by a table look-up and an addition has also proposed [29]. In this dissertation, another new approach to reduce the computation time of division by convergence is proposed.

If Goldschmidt division can be implemented with faster than quadratic convergence, then either the size of the reciprocal table or the number of iterations can be reduced. However, if the cubic convergence algorithm requires additional complex arithmetic computations like sequential multiplications, two consecutive quadratic convergence iterations may be a better choice. Pipelined

multiplier architectures allow one quadratic convergence step to be computed for almost the same amount of time as for a single multiplication. Therefore, cubic convergence algorithms that require complex arithmetic computations have not been practical.

The research focuses on improving the convergence rate of Goldschmidt division without using complex arithmetic. The problem is that the realization of cubic convergence seems to require additional complex arithmetic operations. In this research, a method that solves this problem and achieves near cubic convergence is proposed. The DFQC method (Division with Faster than Quadratic Convergence) is implemented with simple logic circuits to minimize the logic delay. Before presenting the new method, the conventional division-by-convergence algorithms with quadratic convergence algorithm and higher order convergence are presented in Section 4.2. In Section 4.3, the proposed method is explained. In Section 4.4, the detailed implementation of the proposed method is explained. Finally, the simulation results and the effectiveness of the method are summarized in Section 4.5.

4.2 Division with Faster Convergence

Before explaining the new DFQC method, the quadratic convergence algorithm and a faster convergence algorithm for Goldschmidt division are presented.

4.2.1 Quadratic Convergence

The division algorithms explored in the proposed method are various forms of Goldschmidt division. Division can be written as $Q = N/D$ where Q is the quotient, N is the numerator, and D is the denominator. As shown in Section 2.1, the factor F_i and Q_i at the $(i+1)$-th iteration are computed as follows:

$$F_i = (2 - D_{i-1}) = 1 + \epsilon_0^{2^{i-1}} \text{ for } i > 0 \tag{4.1}$$

$$Q_i = \frac{N_i}{D_i} = \frac{N_{i-1}F_i}{D_{i-1}F_i} = \frac{N_{i-1}(1 + \epsilon_0^{2^{i-1}})}{(1 - \epsilon_0^{2^i})} \tag{4.2}$$

As the iteration continues, N_i will converge toward Q with ever-greater precision. Since the error decreases by $\epsilon_0^{2^i}$ quadratically as shown in Equation (4.2), the convergence order of the Goldschmidt division is quadratic.

4.2.2 Division with Faster Convergence

Since the convergence order is determined by F_i in Equation (4.1), faster convergence can be realized by manipulating F_i. As a generalized faster convergence for the Goldschmidt division, F_i with convergence order q is suggested as follows:

$$F_i = \sum_{j=0}^{q-1} \left(\epsilon_0^{q^{i-1}}\right)^j \text{ for } i > 0, q \geq 2 \tag{4.3}$$

where $\epsilon_0 = (1 - D_0)$. If q is 2, F_i in Equation (4.3) becomes the equation for quadratic convergence as shown in Equation (4.1). D_i for the faster conver-

gence is computed using F_i as follows:

$$\begin{aligned} D_i &= D_{i-1} \times F_i \\ &= (1-\epsilon_0^{q^{i-1}})(1+\epsilon_0^{1\cdot q^{i-1}}+\cdots+\epsilon_0^{(q-1)\cdot q^{i-1}}) \\ &= 1-\epsilon_0^{q\cdot q^{i-1}} = 1-\epsilon_0^{q^i} \end{aligned} \tag{4.4}$$

Since the error in D_i decreases by $\epsilon_0^{q^i}$ as shown in Equation (4.4), the convergence order for this Goldschmidt division is q. For example, evaluate Equation (4.4) with $q = 4$ and $i = 1$. This clearly shows that D_1 converges with order 4 as follows:

$$\begin{aligned} F_1 &= 1+\epsilon_0+\epsilon_0^2+\epsilon_0^3 \\ D_1 &= D_0 \times F_1 \\ &= (1-\epsilon_0)(1+\epsilon_0+\epsilon_0^2+\epsilon_0^3) = 1-\epsilon_0^4 \end{aligned}$$

The Goldschmidt division with faster convergence requires additional complex computations, which is a primary problem for the implementation. F_i in Equation (4.3) can be rewritten as follows:

$$\begin{aligned} F_i &= \underbrace{1+\epsilon_{i-1}}_{quadratic} + \underbrace{\sum_{j=2}^{q-1}(\epsilon_{i-1})^j}_{additional} \\ &\text{where } \epsilon_{i-1} = \epsilon_0^{q^{i-1}}, i \geq 1, q \geq 3 \end{aligned} \tag{4.5}$$

The additional computations in Equation (4.5) require powerings and additions. Although the powerings can be implemented more efficiently than consecutive multiplications [33], they are still complex arithmetic operations. If F_i

for higher order convergence can be computed simply, the faster convergence algorithm may become practical.

Cubic convergence, which is the simplest form of faster convergence, requires both a squaring and an addition. The F_i for cubic convergence is expressed by evaluating Equation (4.5) with $q = 3$ as follows:

$$F_i = 1 + \epsilon_{i-1} + (\epsilon_{i-1})^2 \quad \text{where } \epsilon_{i-1} = \epsilon_0^{3^{i-1}}, i \geq 1 \tag{4.6}$$

Since F_i for quadratic convergence can be computed simply by one's complement as shown in Equation (2.3), the computation load is negligible. In contrast, the computation load for cubic convergence is much higher than quadratic convergence since it requires a squaring and an addition. If the squaring is computed by a full multiplication, the cubic convergence algorithm will require two dependent multiplications. On the other hand, two consecutive iterations with quadratic convergence require only 12.5% longer time if the pipelined multiplier comprises 4 stages. In other words, the cubic convergence case requires a similar computation time to that of two cycles of the quadratic convergence algorithm. This explains why cubic convergence in Goldschmidt division has not been regarded as practical. Although cubic convergence can reduce the number of iterations or the size of the reciprocal table, the problem of the heavy computation load has to be mitigated.

4.3 The DFQC Method

As a practical alternative to avoid complex computations, a new division method to implement faster than quadratic convergence is proposed. The first requirement for the new method is that it should be computed by simple logic. The new method uses a rough square approximation for this requirement, so its error analysis is presented in Section 4.3.2.

4.3.1 Division Method with Faster than Quadratic Convergence

The proposed new method, named the DFQC method, implements Goldschmidt division with near cubic convergence by introducing a new F_i computation method as shown in Figure 4.1. The basic operation is the same with the conventional Goldschmidt divider except for the squaring units. The major difference between the proposed method and the conventional quadratic convergence method is that the output of each squaring unit is added when F_i is computed in the new method. While there are two squaring units in Figure 4.1, the number of the squaring units is determined based on the number of iterations and the area cost of the DFQC method.

By adopting an approximate squaring and carry-save representation, the DFQC method avoids the complex computations of "true" cubic convergence and achieves faster than quadratic convergence. Although many efficient squaring methods have been suggested, they still require a heavy computation. In contrast, the DFQC method adopts a simple approximate squaring computation. Although approximate squaring does not provide the full cubic

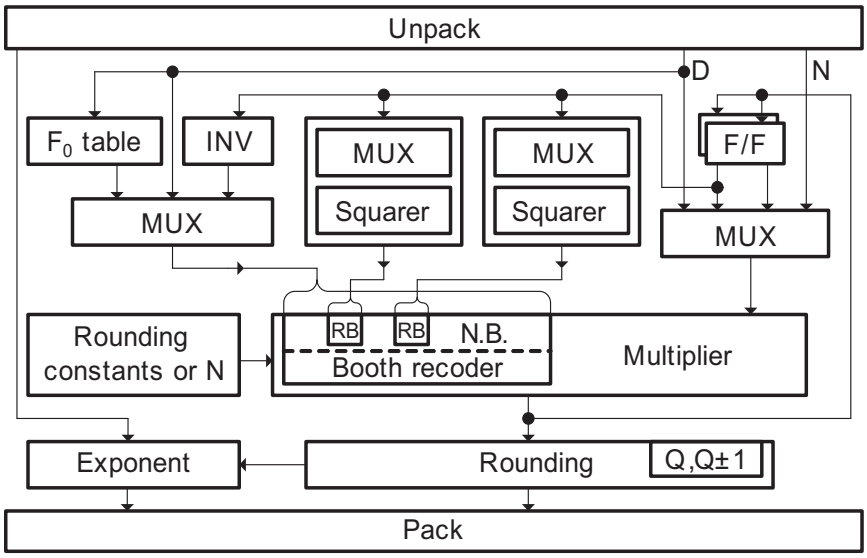


Figure 4.1: A floating-point Goldschmidt divider implemented using the DFQC method with two squaring units.

convergence, the convergence speed is faster than quadratic. In addition to the approximate squaring, the addition is converted into carry-save representation and is merged into the redundant binary Booth recoders (RBBR) [34] of the multiplier. Therefore, the addition in the DFQC method requires no real adder. RB and NB in Figure 4.1 stand for RBBR and NBBR (normal binary Booth recoders). As shown in Figure 4.1, RBBRs are used only where the squaring units are interfaced to in order to save the area.

The detail algorithm of the DFQC method, which computes an approximation $F_i^{'}$ to F_i through a one's complement, an approximate squarer, and a redundant binary Booth recoder, is shown in Figure 4.2. In Figure 4.2 and the rest of this chapter, $F_i^{'}$, $N_i^{'}$, $D_i^{'}$ and $\epsilon_i^{'}$ means the variables computed by

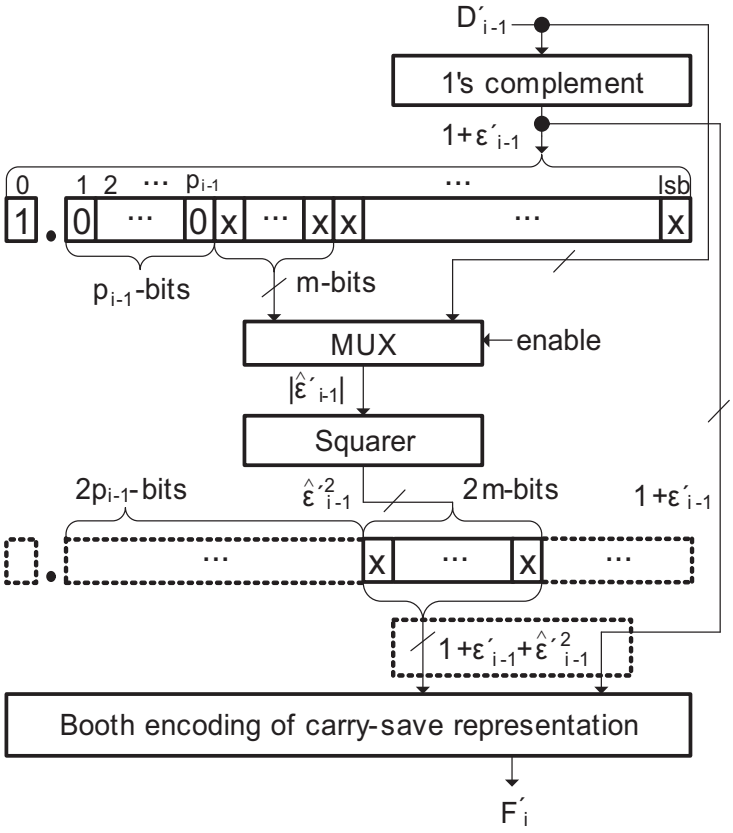


Figure 4.2: $F_i^{'}$ computation using $m \times m$ squaring for near cubic convergence.

the DFQC method, which correspond to F_i, N_i, D_i and ϵ_i for the conventional Goldschmidt division. p_{i-1} is defined as the minimum number of consecutive zeros/ones following the binary point in $(1+\epsilon_{i-1}^{'})$. The initial p_0 is determined by the accuracy of the reciprocal table for $F_0^{'}$, and the other p_{i-1} is determined by the performance analysis result in Section 4.3.2. Since each p_{i-1} is determined in advance, a leading zero/one detector unit and a left shifter are not required. m is the precision of the approximate squarer input.

In the DFQC method, $N_i^{'}$ and $D_i^{'}$ is also computed using $F_i^{'}$ via iterations as the conventional method. $D_{i-1}^{'}$ in Figure 4.2 is $(1-\epsilon_{i-1}^{'})$, which comes from the previous iteration. $(1+\epsilon_{i-1}^{'})$ is computed by forming the one's complement. The input of the squarer is defined as $\hat{\epsilon}_{i-1}^{'}$, which is an m-bit string from $(p_{i-1}+1)$-th bit to $(p_{i-1}+m)$-th bit of $(1+\epsilon_{i-1}^{'})$ in Figure 4.2. If $\epsilon_{i-1}^{'}$ is negative, the m-bit string is inverted before squaring in order to input $|\epsilon_{i-1}^{'}|$ to the squarer. Since the rough approximate value of $(\epsilon_{i-1}^{'})^2$ is computed using $\hat{\epsilon}_{i-1}^{'}$, the output, $\hat{\epsilon}_{i-1}^{'2}$, is a $2m$-bit string.

The output of the squaring unit $(\hat{\epsilon}_{i-1}^{'2})$ and $(1+\epsilon_{i-1}^{'})$ form $F_i^{'}$ through the Booth recoding for carry-save representation as follows:

$$F_i^{'} = (1+\epsilon_{i-1}^{'}) + \hat{\epsilon}_{i-1}^{'2} \quad \text{for } i > 0 \tag{4.7}$$

Since the two numbers are interpreted as a carry-save representation, the redundant binary Booth recoder adds the carry-save representation with no explicit addition. The msb bit of the squarer output is located at the $(2p_{i-1}+1)$-th bit of $(1+\epsilon_{i-1}^{'})$ in the redundant Booth binary recoder. The detailed implementation is presented in Section 4.4.

4.3.2 Performance Analysis of the DFQC Method

The performance of the DFQC method is evaluated by determining the deviation from the true quotient (i.e., the error) at each iteration step. It is a result of (a) the approximate squaring and (b) the truncation to a limited multiplier precision. The approximate squaring determines the speed

of the convergence in the DFQC method. On the other hand, the truncation error due to the limited multiplier precision is too small compared to the error of the approximate squaring since the maximum effective p_{i-1} is around a quarter of the multiplier precision in the DFQC method. In addition, this truncation error will be ignored by increasing the required multiplier precision by several extra bits [23], which is determined in Section 4.5.2. Therefore, the truncation error due to the limited multiplier precision is not considered in the performance analysis.

The relative error term e_i [22] is used as the performance metric to evaluate the speed of the convergence for the DFQC method. Since the truncation error due to a limited multiplier precision can be ignored for this performance evaluation, it is assumed that $Q = N/D = N_i^{'}/D_i^{'}$. Since the approximate quotient is $N_i^{'}$, the absolute error $|N_i^{'} - Q|$ and the relative error term e_i of the approximate quotient after the i-th iteration are as follows:

$$\begin{aligned} |N_i^{'} - Q| &= \left| N_i^{'} - \frac{N}{D} \right| = \left| N_i^{'} - \frac{N_i^{'}}{D_i^{'}} \right| \\ &= \left| \frac{N_i^{'}(D_i^{'} - 1)}{D_i^{'}} \right| \\ &= |Q \cdot (D_i^{'} - 1)| \end{aligned}$$

$$e_i = \frac{|N_i^{'} - Q|}{Q} = |D_i^{'} - 1|$$

Due to the floating-point normalization process, the maximum errors for both Q ranges, $0.5 \leq Q < 1$ and $1 \leq Q < 2$, are required. Since the analyzed maximum e_i is independent of Q, it can be converted easily into the maximum

absolute error by multiplying it by Q. In contrast, the analyzed maximum absolute error is inconvenient for this purpose. In addition, the relative error definition is also used for quadratic convergence [22] since it shows that $e_{i+1} = e_i^2$. Accordingly, the relative error term e_i is adopted as the performance metric to evaluate the DFQC method here.

The performance of the DFQC method is evaluated by a maximum relative error analysis considering the approximate squarer. Before evaluating e_i, the error of the approximate squaring is determined. $\tilde{p}_{i-1}$ is defined as a real value that satisfies

$$2^{-\tilde{p}_{i-1}} = max|D_{i-1}^{'} - 1| = max|\epsilon_{i-1}^{'}| \ .$$

p_{i-1} can be also defined as $\lfloor \tilde{p}_{i-1} \rfloor$. Therefore, $\epsilon_0^{'}$ is guaranteed to be less than 2^{-p_0}. p_{i-1} and $\tilde{p}_{i-1}$ are expressed as p and $\tilde{p}$ to simplify the equations. Since the maximum error due to truncation at the input to the m-bit squaring in Figure 4.2 is $-2^{-(p+m)}$, the maximum squaring output error (E_{sq_max}) is

$$\begin{aligned} E_{sq\_max} &= \epsilon_{i-1}^{'2} - min(|\hat{\epsilon}_{i-1}^{'}|^2) \\ &= \epsilon_{i-1}^{'2} - (|\epsilon_{i-1}^{'}| - 2^{-(p+m)})(|\epsilon_{i-1}^{'}| - 2^{-(p+m)}) \\ &= 2^{-(p+m-1)} \cdot |\epsilon_{i-1}^{'}| - 2^{-2(p+m)} \\ &= 2^{-p-\tilde{p}-m+1} - 2^{-2(p+m)} \ . \end{aligned}$$

If $\epsilon_{i-1}^{'}$ is assumed to be positive, the maximum relative error of the quotient

(e_{i_max}) due to E_{sq_max} is

$$\begin{aligned} e_{i\_max} &= max|D_i^{'} - 1| \\ &= |(1 - \epsilon_{i-1}^{'})(1 + \epsilon_{i-1}^{'} + \epsilon_{i-1}^{'2} - E_{sq\_max}) - 1| \\ &= |(1 - \epsilon_{i-1}^{'})(1 + \epsilon_{i-1}^{'} + \epsilon_{i-1}^{'2} - 2^{-p-\tilde{p}-m+1} + 2^{-2p-2m}) - 1| \\ &= |\epsilon_{i-1}^{'3} + (1 - \epsilon_{i-1}^{'})(2^{-p-\tilde{p}-m+1} - 2^{-2p-2m})| \ . \end{aligned}$$

Since $max(\epsilon_{i-1}^{'})$ is $2^{-\tilde{p}}$, the maximum relative error of the quotient at the i-th iteration is

$$\begin{aligned} e_{i\_max} &= |2^{-3\tilde{p}} + (1 - 2^{-\tilde{p}})(2^{-p-\tilde{p}-m+1} - 2^{-2p-2m})| \\ &< |2^{-p-\tilde{p}-m+1} - 2^{-2p-2m} + 2^{-3\tilde{p}}| \\ &< |2^{-p-\tilde{p}-m+1} + 2^{-3\tilde{p}}| \end{aligned} \tag{4.8}$$

where $i > 0$. Even if $\epsilon_{i-1}^{'}$ is negative, the maximum error in Equation (4.8) is still correct. If m is larger than $p+1$, the error will not be dominated by m as shown in Equation (4.8). Therefore, it is reasonable to assume that $m \leq p+1$.

The errors of the quotient for the conventional quadratic convergence and the DFQC method are compared using a 5-bit-in 5-bit-out reciprocal ROM table as shown in Table 4.1. In order to evaluate the errors, $\tilde{p}_0$ has to be determined. Since $0.5 \leq D < 1$, the error of an optimized reciprocal table [16] corresponding to input x is

$$2^{-k}(\frac{1}{4x^2} + \frac{1}{2^{g+1}}) \tag{4.9}$$

where $0.5 \leq x < 1$, k is the number of input index bits, and $k + g$ is the bit width of the table output. Since $\epsilon_0^{'}$ is dependent on only the error of the

reciprocal ROM table, $\epsilon_0^{'}$ is computed using Equation (4.9) as follows:

$$\epsilon_0^{'} = \epsilon_0 = (1 + \frac{1}{2^{g+1}})2^{-(k+1)} \tag{4.10}$$

Since $\epsilon_0^{'}$ is computed as $2^{-5.4}$ in a 5-bit-in 5-bit-out reciprocal ROM table case, $\tilde{p}_0$ before the first iteration is 5.4. For the DFQC method, e_{i_max} is calculated by evaluating Equation (4.8) with $\tilde{p}_0 = 5.4$ as shown in Table 4.1. If $2p_{i-1}$ is beyond the multiplier precision, $F_i^{'}$ cannot be compensated using the multiplier. Since double precision floating-point division is assumed in Table 4.1, the first two iteration steps are computed by the DFQC method, but the third step is computed by the conventional quadratic convergence algorithm.

The performance of the DFQC method can also be evaluated by the speed of the convergence, which is expected to be faster than quadratic. If $-2^{-3\tilde{p}}$ in Equation (4.8) is ignored, the error at each iteration step can be formulated simply. Since m is less than p_0 in many practical cases, this assumption is reasonable. As a result, e_{i_max} in Equation (4.8) can be expressed

Table 4.1: Maximum errors of the conventional and the DFQC method ($\tilde{p}_0 = 5.4, m = 4$)

Method	$i = 0$	$i = 1$	$i = 2$	$i = 3$
Quadratic $\lvert D_i - 1\rvert$	$2^{-5.4}$	$2^{-10.8}$	$2^{-21.6}$	$2^{-43.2}$
DFQC $\lvert D_i^{'} - 1\rvert$	$2^{-5.4}$	$2^{-13.2}$	$2^{-29.2}$	$2^{-58.4}$

simply as follows:

$$e_{i\_max} = max|D_i' - 1| \approx 2^{-(2^i \cdot (p_0+m-1)-m+1)} \tag{4.11}$$

On the other hand, $max|D_i - 1|$ for the current quadratic convergence algorithm is $2^{-2^i \cdot p_0}$. By comparing these two errors, it is clear that the DFQC method has faster than quadratic convergence.

4.4 Implementation Details of the DFQC Method

The key elements of the DFQC method are the efficient hardware implementation for the combination of approximate squarers and Booth recoders for carry-save representation. In this section, the detailed implementation of each unit is explained at the gate level.

4.4.1 Squarer

An m-bit squarer can be implemented with simple logic by optimizing a parallel multiplier algorithm if m is small. To reduce the logic delay, both the carry save adder tree and the final stage adder are merged into one logic circuit. Although a large squarer would speed the convergence, the delay and the complexity increase significantly, so a 4-bit squarer is used for the DFQC method. Assume that $X[i:j]$ represents a bit string from i-th bit to j-th bit of X, and the i-th bit is associated with 2^{-i}. Since the input of the 4-bit squarer is

$$s_{in} = a \cdot 2^{-p-1} + b \cdot 2^{-p-2} + c \cdot 2^{-p-3} + d \cdot 2^{-p-4} \ ,$$

the squarer after the optimization is implemented as follows:

$$s_{out}[2p+1:2p+8] = \{ab, a(\bar{b}+c), abd + c(a \oplus b), \\ b\bar{c}\bar{d} + d(a \oplus b), d(b \oplus c), c\bar{d}, 0, d\}$$

If a squarer output is not used for an iteration or the multiplier is operating for general use, the outputs of the squarer are turned off by the multiplexers with enable pins in Figure 4.3. While a 3-input multiplexer where one input is always 0 can be used, a 2-input multiplexer with an enable pin is more efficient.

4.4.2 Limitation of the DFQC Method in Radix-4 Multiplier

The DFQC method is very effective when division is implemented using a radix-8 multiplier, which may already exist for multiplication in the system. Since the computation of the squarer and the RBBR for the DFQC method is performed in parallel with the 3X adder in a radix-8 multiplier, the DFQC method may not be on the critical timing path of the multiplier. However, if the system has a radix-4 multiplier, the DFQC method may increase the clock cycle due to the delay of the DFQC method since the squarer and the radix-4 RBBRs may be on the critical timing path. In this case, the iteration step using the DFQC method needs one additional clock, and the other steps can be computed using NBBRs. If one additional clock is allowed, it is possible that the speed of the convergence becomes faster by increasing the size of the squarer, m. Although the DFQC method has drawbacks in a system that has

a radix-4 multiplier, it still speeds the convergence and reduces the required precision of the initial approximation.

Radix-8 multipliers are used mostly for many floating-point units in spite of their complexity [3, 35, 36] because radix-8 Booth recoders are effective to reduce the delay due to wire congestion in deep submicron VLSI technology. In addition, the radix-4 RBBRs can be implemented using a similar architecture. Therefore, this research focuses on the DFQC method with a radix-8 multiplier.

4.4.3 Radix-8 Redundant Binary Booth Recoder for Carry-Save Representation

The squarer output, $\hat{\epsilon}^2$, and $(1 + \epsilon)$ are added without a real adder inside the Booth recoding part of a multiplier. The two binary numbers form a carry-save representation, and then the carry-save representation is converted into radix-8 Booth recoded digits by a redundant binary Booth recoder (RBBR) [34]. Inside the RBBR, the carry-save representation is converted into a signed digit [37] representation (sign bits and magnitude bits). This signed digit representation is divided into small groups that do not generate group carry values that are greater than 1. The output signals of the Booth recoder are comprised of a sign bit and signals for 0, ± 1, ± 2, ± 3, and ± 4. These five output signals are suitable for the current multiplier hardware architectures to reduce the wire congestion.

The radix-8 RBBR circuit implemented in this dissertation is modified

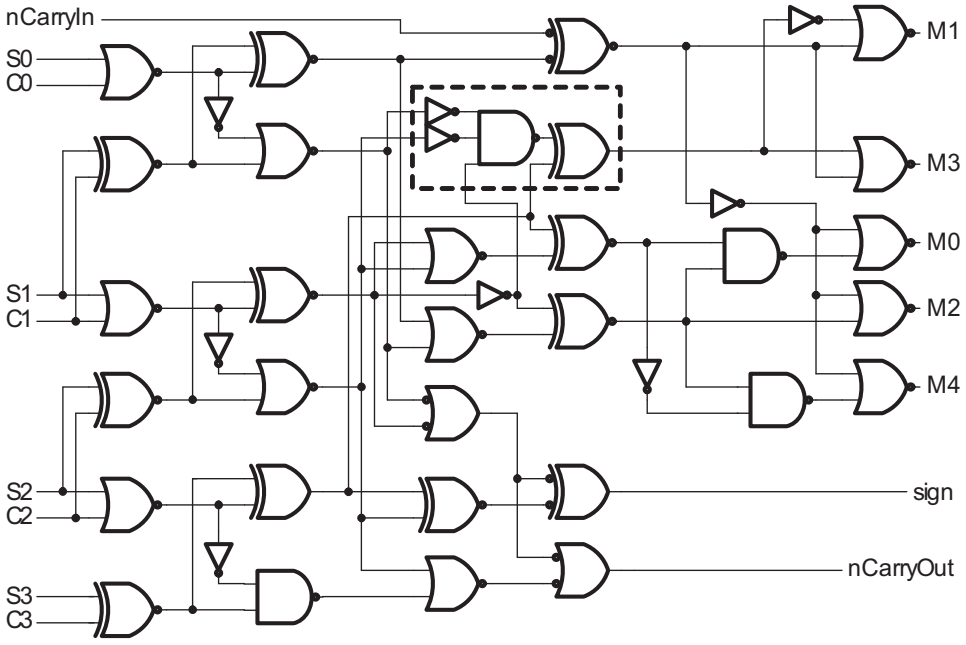
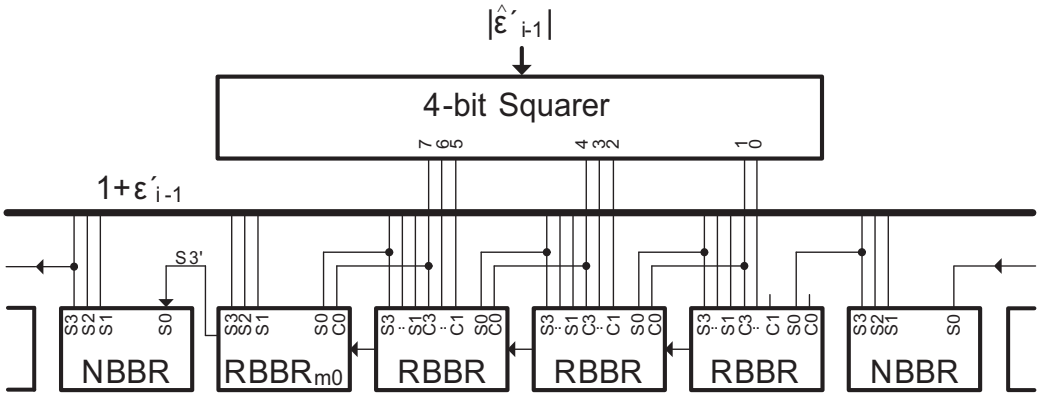


Figure 4.3: Modified radix-8 carry-save Booth recoder. The $M1/M3$ selection logic is added as shown in the dotted box.

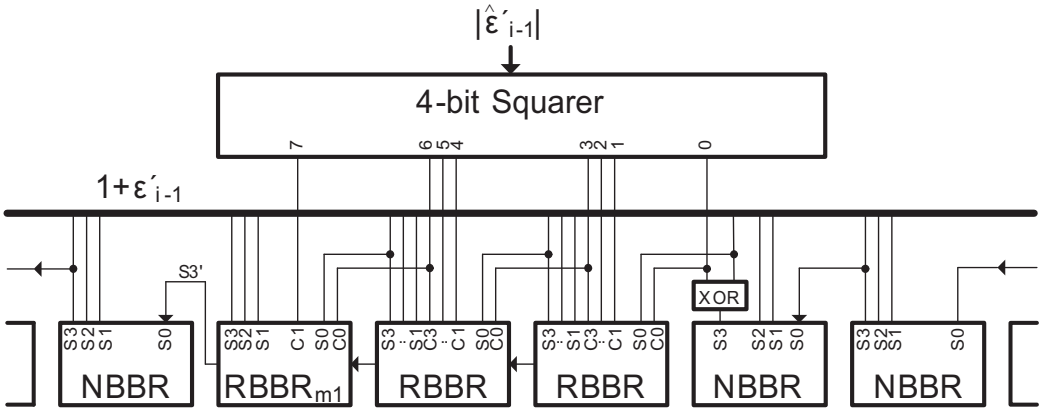
slightly from the original radix-8 RBBR circuit [34] as shown in Figure 4.3 in order to operate with carry-save encoded inputs. If the radix-8 RBBR for carry-save input is implemented with the preprocessing scheme from [34], the $M1$ and $M3$ output signals may be driven incorrectly. The problem occurs if a carry-save encoded input makes an intermediate signed digit combination like $\times 1\bar{1}$. Here this problem is resolved by adding $M1/M3$ selection logic. In addition, a minor typographical error on a wire connection in the original circuit diagram has been corrected. For further optimization, negative logic is used, and the inverters in the RBBR have been reallocated for speed and area.

4.4.4 Special RBBRs to interface with NBBRs

The 4-bit squarer outputs are connected to either four or three radix-8 RBBRs as shown in Figure 4.4. The most significant RBBR that is labeled as $\mathrm{RBBR_{m0}}$ in Figure 4.4(a) is a special RBBR that does not generate an explicit



(a)



(b)

Figure 4.4: 4-bit squarer and radix-8 redundant binary Booth recoders. (a) Four RBBR case, (b) Three RBBR case.

carry out signal, so a NBBR can be connected to the left side of the $\mathrm{RBBR_{m0}}$. Instead, an output $S_3{}'$ from the $\mathrm{RBBR_{m0}}$ is connected to the next NBBR. The output $S_3{}'$ is implemented as follows:

$$S_3{}' = S_3 + S_1S_2(S_0 + C_0)$$

Since C_3 and C_2 of the $\mathrm{RBBR_{m0}}$ are always zero, $S_3{}'$ takes the role of a carry out signal. In this case, the 4-bit squarer requires three normal RBBRs and a special RBBR. Due to the special RBBR, the normal binary Booth recoders are used where a squarer is not connected, and the number of RBBRs can be minimized.

If only the msb bit of the squarer outputs is connected to the RBBR as shown in Figure 4.4(b), another special RBBR that is labeled as $\mathrm{RBBR_{m1}}$ can be used to reduce the number of RBBRs. In this case, $S_3{}'$ of the $\mathrm{RBBR_{m1}}$ is implemented as follows:

$$S_3{}' = S_3 + S_2S_1C_1 + S_2(S_1 + C_1)(S_0 + C_0)$$

Since the delay of an NBBR is shorter than that of an RBBR, the delay of $S_3{}'$ does not create a critical timing path. In addition to the $\mathrm{RBBR_{m1}}$, the lsb bit of the squarer outputs is connected to the NBBR through an XOR gate. The msb input of the NBBR is:

$$S_3 = S_{3in} \oplus C_{3in}$$

where S_{3in} is from $(1 + \epsilon'_{i-1})$, and C_{3in} is the lsb bit of the squarer outputs. In this case, the lsb bit of the squarer outputs is connected to the NBBR instead

of the RBBR, so the 4-bit squarer requires an $RBBR_{m1}$, two normal RBBRs, and an XOR gate.

4.5 Simulation and Results

4.5.1 Simulation Environment

An example double precision floating-point Goldschmidt divider as described here has been developed using SystemC 2.2.0 and Verilog HDL. A SystemC model was developed to check the validity of the DFQC method and the exhaustive verification. Although a high-level SystemC model is very useful for algorithm verification due to its fast simulation speed, it is not adequate for the evaluation of the delay and area costs. Therefore, a Verilog model based on the SystemC model has been also developed to evaluate the delay and area of the DFQC method. The Verilog model has been synthesized using Synopsys design compiler (A-2007.12-SP4) and the 0.18μm OSU-stdcells-TSMC018 library.

4.5.2 Implementation of an Example Goldschmidt Divider

The example Goldschmidt divider uses a 59×59-bit multiplier, a 5-bit-in 5-bit-out reciprocal table, and a 4-bit squarer. It also supports the four IEEE-754 rounding modes by computing the sign of the remainder [3, 9, 10], and it performs 3 iterations after the normalization of the denominator.

If the four IEEE rounding modes and an additional guard bit for the normalization process are considered, the final quotient error must be less than

$\frac{1}{4} \cdot 2^{-53}$ [9]. The quotient error at the final iteration step is due to the finite number of iterations and the limited multiplier precision. The error due to the finite iterations, e_{3_max}, can be reduced by enlarging the reciprocal ROM table. Therefore, it is assumed that an appropriate reciprocal ROM table guarantees e_{3_max} less than 2^{-lsb}, the lsb bit of a multiplier. The other cause of the error, limited multiplier precision, can be reduced by increasing the precision of the multiplier. Since $F_i^{'}$ is computed partially by a one's complement operation, the maximum absolute error in the final quotient [31] is

$$\begin{aligned} max|E| &= max|N_i^{'} - Q| \\ &= \left| -(\frac{5}{2}N_{2max}^{'} + 2) \cdot 2^{-lsb} - N_{3max}^{'} \cdot e_{3\_max} \right| \\ &\simeq |-4.5 \cdot 2^{-lsb} - 1 \cdot 2^{-lsb}| \\ &\simeq 5.5 \cdot 2^{-lsb} < 6 \cdot 2^{-lsb} \end{aligned} \tag{4.12}$$

where $Q < 1$. When $Q \geq 1$, the effective error is halved due to the floating-point normalization, so the worst case error occurs when $Q < 1$. As a result, $6 \cdot 2^{-lsb}$ must be less than $\frac{1}{4} \cdot 2^{-53}$, and the width of the multiplier has to be at least 59-bits.

The reciprocal ROM table data have to be optimized tightly in order to limit the reciprocal error as shown in Equation (4.10). Since the range of D is $0.5 \leq D < 1.0$, $D[0:1]$ is always `01`, and the input of the table is $D[2:(2+k-1)]$. $F_0[0]$ on the table output is always `1` even if $D = 0.5$. Since each reciprocal value in the table must produce the minimum error in an interval, $[x, x + 2^{-(k+1)})$, the k-bit-in $(k+g)$-bit-out reciprocal ROM table

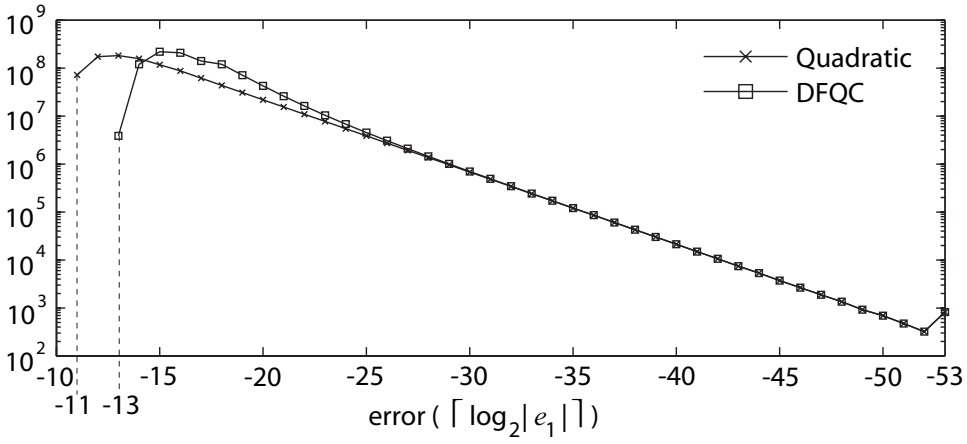
is computed using mid-point values as follows:

$$r(x) = \frac{1/x + 1/(x + 2^{-(k+1)})}{2} + 2^{-(k+g+1)}$$
$$\text{for } F_0(x) = r(x)[0, k+g],\ x = D[0, k+1]$$

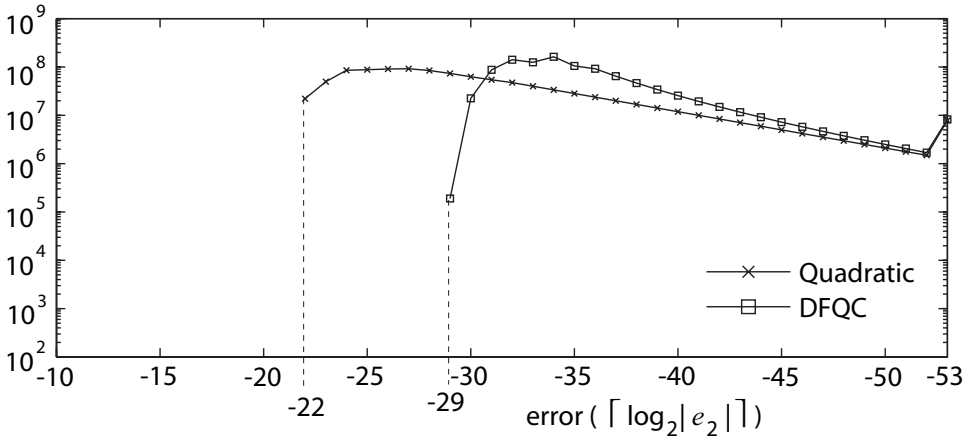
4.5.3 Simulation for the DFQC Performance Analysis

To confirm correct operation of the DFQC method and to validate the error analysis, 10^9 random double precision floating-point test vectors were used in a simulation using a SystemC model. The simulations were performed for two cases. One is the DFQC method with a 5-bit-in 5-bit-out reciprocal table ($p^{'} = 5.4$) and a 4-bit squarer, and the other is for the standard quadratic convergence algorithm with the same conditions. The error histograms of e_i after the 1_{st} iteration and the 2_{nd} iteration are shown in Figure 4.5. The histograms show the distribution of $\lceil \log_2(e_i) \rceil$ after each iteration step.

The error histograms in Figure 4.5 show the effectiveness of the DFQC method. The iteration errors are bounded as predicted by the performance analysis in Section 4.3.2. The maximum errors of the DFQC method are much smaller than those of the quadratic convergence. The first non-zero value for the first iteration in Figure 4.5(a) starts at -13, which means that e_{1_max} of the DFQC method is smaller than 2^{-13} as expected from the error analysis. Figure 4.5(b) shows that e_{2_max} for the DFQC method is smaller than 2^{-29}.



(a)



(b)

Figure 4.5: Error histograms of the DFQC method and the quadratic convergence ($p^{'} = 5.4, m = 4$). (a) After the 1_{st} iteration, (b) After the 2_{nd} iteration.

4.5.4 Verification of Division by the DFQC Method

In addition to checking the intermediate results during the division as shown in Section 4.5.3, the final results of the double precision floating-point division are also checked for all four IEEE rounding modes.

The verification of the Goldschmidt division with the DFQC method consists of two parts. First, the final division results of the four IEEE rounding modes were checked by performing 10^9 divisions using random double precision floating-point numbers. Second, the final results were also checked by performing divisions with all possible 17-bit precision fixed-point numerators and denominators (a total of 2^{32} test vectors) to insure that no special cases were missed by the random test vectors. Since exhaustive verification using double precision floating-point vectors was not feasible, the exhaustive 17-bit precision test was a practical alternative. For the 17-bit significand simulation, a 23×23-bit multiplier, a 4-bit-in 4-bit-out reciprocal table, and a 3-bit squarer were implemented, and 2 iterations were performed. Since the X87 FPU does not support 17-bit precision division, a digit recurrent divider was also implemented as a reference to verify the results.

Also the maximum absolute errors in the final quotient have been checked via simulation. The absolute error of the final quotient in the simulation (Q_A) is computed using the remainder equation as follows:

$$E = Q_A - Q = \frac{Q_A \times D - N}{D} \tag{4.13}$$

Table 4.2 shows the maximum absolute errors during the simulation, which

Table 4.2: Maximum approximation errors of the final quotients during the simulation

Error type	10^9 random numbers		exhaustive test vectors	
	$Q < 1$	$Q \geq 1$	$Q < 1$	$Q \geq 1$
E_{max+}	$2 \cdot 2^{-58}$	$2 \cdot 2^{-58}$	$2 \cdot 2^{-22}$	$1 \cdot 2^{-22}$
E_{max-}	$-4 \cdot 2^{-58}$	$-6 \cdot 2^{-58}$	$-4 \cdot 2^{-22}$	$-6 \cdot 2^{-22}$

include margins to compensate for the truncation error in Equation (4.13). All the errors are bounded inside the maximum allowable error [9], $\frac{1}{4} \cdot 2^{-n}$ where $n = 53$ or 17.

If Q is greater than 1 in Table 4.2, the effective error is halved. In the case $Q > 1$, Q_A is normalized, and the $(n-1)$-th bit becomes a *ulp* instead of the n-th bit. Thus, the n-th bit serves as an additional bit, which further reduces the error. For example, although E_{max-} for $Q > 1$ is shown as $-6 \cdot 2^{-lsb}$ in Table 4.2, the effective error after normalization is $-3 \cdot 2^{-lsb}$.

4.5.5 Delay for the DFQC Method

The delay of the DFQC method at the first pipeline stage of a 59×59-bit multiplier was evaluated using Synopsys design compiler (A-2007.12-SP4) with the 0.18μm OSU-stdcells-TSMC018 library. It is known that the 3X adder is on the critical path in a radix-8 multiplier [38]. The 4-bit squarer and the redundant binary Booth recoder (RBBR) can compute in parallel with the 3X adder. If the squarer and RBBRs compute during the 59-bit 3X adder

Table 4.3: Critical path delays for the DFQC method and a 3X adder

Functional block	Delay (ns)
DFQC: 4-bit squarer with 1's comp., RBBR	1.59
- MUX & 4-bit squarer with 1's comp.	0.75
- Radix-8 redundant binary Booth recoder	0.84
Quadratic: 3X adder	1.65

computation, the squarer and RBBRs will not affect the total pipeline cycle time. The simulation shows that the 3X adder is on the critical timing path in the example double precision floating-point Goldschmidt divider. The delays of the squarer, the RBBR [34], and a 3X adder [38] are shown in Table 4.3. The 3X adder has a bit more delay than the squarer and RBBR. In addition, there are relatively long interconnects inside the 59-bit 3X adder in contrast to the squarer and the RBBR. Therefore, the delay of the squarer and RBBR are expected to be less than that of the 3X adder after the interconnect capacitances are considered. On the other hand, if the Booth multiplexers move to the second stage due to the large number of flip-flips as in the S/390 processor [39], neither the delay of the squarer and RBBR nor the delay of the 3X adder will be on the critical path. In conclusion, this critical timing path analysis shows the feasibility of the DFQC method from a delay perspective.

4.5.6 Area of the DFQC Method

The table specifications required for the double precision floating-point division are determined by e_{i_max} of the last iteration. Since e_{i_max} of the last iteration has to be less than 2^{-58} to implement the double precision floating-point division with IEEE rounding, the dimensions of reciprocal ROM tables can be calculated using Equation (4.8) and Equation (4.10) as shown in Table 4.4. For example, the DFQC method for double precision floating-point division requires a 13-bit-in 13-bit-out table for 2 iterations.

Since the DFQC method requires additional logic to reduce the required table size, an analysis of the area of the new method is necessary. The DFQC method reduces the table size as shown in Table 4.4. The additional logic area is estimated based on the synthesis result using the design compiler and the 0.18μm OSU-stdcells-TSMC018 library as shown in Table 4.5. The ROM table area estimation is based on an experimental result [28] that an area for a ROM table with 7–11 inputs corresponds to 35 full adders/Kbit and the area of a full adder is 9 NAND gates.

Table 4.4: Initial errors and the required table sizes for double precision floating-point

Method	3 iterations A		3 iterations B*		2 iterations	
	ϵ_0	table	ϵ_0	table	ϵ_0	table
Quadratic	$2^{-7.4}$	$2^7 \times 7$	$2^{-7.4}$	$2^7 \times 7$	$2^{-14.6}$	$2^{14} \times 15$
DFQC	$2^{-5.4}$	$2^5 \times 5$	$2^{-6.4}$	$2^6 \times 6$	$2^{-13.4}$	$2^{13} \times 13$

*The DFQC is applied only at the first iteration.

Table 4.5: Area costs of the DFQC method and the quadratic method (OSU-stdcells-TSMC018)

Method	ROM	Mux & Squarer	RBBR overhead	Total (μm^2)
Quad. / 2 itr.	196,560 [a]	-	-	196,560
DFQC/ 2 itr.	143,640 [a]	763	2,320 [b]	146,723
Quad. / 3 itr.	6,615	-	-	6,615
DFQC/ 3 itr. A	1,181	1,526	4,640 [b]	7,347
DFQC/ 3 itr. B	2,835	763	1,836 [b]	5,434

[a] Areas using the bipartite ROM table[24].

[b] NBBR:609, RBBR:1,233, $\mathrm{RBBR_{m0}}$:1,057, $\mathrm{RBBR_{m1}}$:1,141

In the 2-iteration Goldschmidt division, the DFQC method reduces the effective table area by 25.4% (i.e., from 196,560μm^2 to 146,723μm^2) after considering the increased logic. The DFQC method requires a 4-bit squarer, three radix-8 RBBRs, and an $\mathrm{RBBR_{m0}}$. Since a large ROM table is typically implemented by the bipartite ROM table method, it is assumed that a $2^{14} \times 15$ ROM table is implemented using a $2^{10} \times 17$ table and a $2^{10} \times 9$ table, and a $2^{13} \times 13$ ROM table is implemented using a $2^{10} \times 15$ table and a $2^9 \times 8$ table.

In the 3-iteration Goldschmidt division, the DFQC method reduces the effective table area by 17.9% (i.e., from 6,615μm^2 to 5,434μm^2) as shown in the 3 iterations B case in Table 4.5. In this case, the DFQC method is used only for the first iteration, and it requires a 4-bit squarer, two radix-8 RBBRs, an $\mathrm{RBBR_{m1}}$, and an XOR gate. On the other hand, if the DFQC method

is used for both the first and second iterations, the logic increase due to the DFQC method is larger than the amount of the table reduction as shown in the 3 iterations A case in Table 4.5. The analysis result shows that it is effective to apply the DFQC method only at the first iteration in the case of the 3-iteration double precision floating-point Goldschmidt division. Unlike the 2-iteration case, the bipartite ROM table method [24] is not adopted because it has a similar logic complexity as the DFQC method when the ROM is small.

In conclusion, the results show that the DFQC method is an effective method to reduce the size of the reciprocal ROM table in Goldschmidt division.

4.6 Summary

A new method for Goldschmidt division that implements faster than quadratic convergence has been presented. The main contribution of this research is its presentation of methods that realize the near cubic convergence with simple logic circuits and that minimize the logic delay and the area. Although division with cubic convergence has been regarded as impractical, this research shows that the proposed method using modified redundant binary Booth recoders (RBBR) and simple approximate squarers can speed the convergence effectively. The DFQC method can be implemented by minor modifications of the Booth recoders of a multiplier and a simple approximate squarer. Especially, since the special RBBRs with a single carry signal can be interfaced to a NBBR, the number of RBBRs can be minimized, which mitigates the area increase due to the DFQC method. A 4-bit squarer is

adopted for the DFQC method implementation. Due to the redundant binary Booth recoder, the squarer output is merged into the multiplier without a real addition.

The performance analysis and the simulation results in SystemC show that even a 4-bit squarer can speed the convergence effectively. The simulation result using the Verilog implementation also shows that the delay due to the DFQC method does not reduce the feasibility of the new method if a system already has a radix-8 multiplier. The area analysis shows that the effective area for the reciprocal ROM table can be reduced by 25% for the 2-iteration double precision floating-point Goldschmidt division. For the 3-iteration Goldschmidt division, the effective area of the ROM can be reduced by 17%. Due to the increased logic, the DFQC method is more effective in Goldschmidt dividers that have large reciprocal ROM tables. The validity of the DFQC method has been verified through error analysis and extensive simulation. The verifications of the division through the DFQC method have been performed for all 4 IEEE-754 rounding modes using both 10^9 random double precision floating-point test vectors and an exhaustive suite of 17-bit precision test vectors. In conclusion, the DFQC method can be an effective approach to reduce the area of the ROM table for Goldschmidt divisions.

Chapter 5

Goldschmidt Iterative Divider for Quantum-dot Cellular Automata

5.1 Overview

Quantum-dot cellular automata (QCA) [5, 6] is a promising emerging nanotechnology that may mitigate the problems due to the continued scaling of semiconductor feature sizes. Since QCAs operate according to different principles from CMOS technology, they require different design methods.

Iterative computational circuit designs for QCA are difficult to build with conventional sequential circuit design methods that are based on state machines. State machines for QCA have problems due to long delays between the state machine and the units to be controlled. Even a simple 4-bit microprocessor that has been implemented with QCA [40] was done without using a state machine. Due to the difficulty of designing sequential circuits, there has been little research into using QCA to realize iterative computational units, such as dividers. Most previous research has concerned simpler arithmetic unit designs, such as adders and multipliers.

In this chapter, a Goldschmidt iterative divider is designed using a new architecture to solve the difficulty in designing iterative computation units. In

Section 5.2, quantum-dot cellular automata and its operations as logic components are explained, and problems of conventional Goldschmidt division architectures in QCA is presented. In Section 5.3, the proposed method to avoid the problems in the conventional architecture is presented. In Section 5.4, an implementation of the Goldschmidt divider using the proposed method is reviewed in detail. Finally, the simulation result of the design and the summary are presented in Section 5.5 and 5.6.

5.2 Quantum-dot Cellular Automata

5.2.1 QCA Cell

A QCA cell [6] has four quantum dots and two electrons that are trapped inside the dots as shown in Figure 5.1. Binary information is encoded by the positions of the electrons, and a QCA cell allows two available polarizations, $P = \pm 1$. Since the quantum dots are coupled by tunnel barriers,

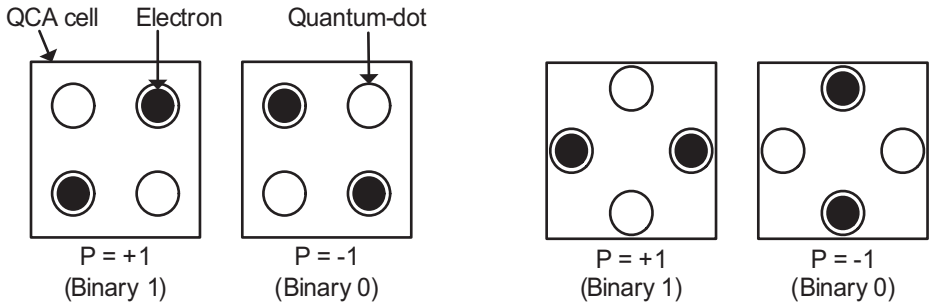


Figure 5.1: Basic QCA cells with two possible polarizations. (a) Regular cells, (b) 45° rotated cells.

the two electrons can change their positions freely by controlling the potential barriers using a clocking mechanism. The computation is performed by interactions based on Coulombic forces between neighboring QCA cells. Since the basic principle of operation is very different from CMOS, QCA has many unique characteristics [41–43].

5.2.2 Logic Gates in QCA

The basic circuit elements in QCA are inverters and majority gates. All the other logic gates, such as AND gates and OR gates, can be realized using these basic elements. A conventional QCA inverter and its symbol are shown in Figure 5.2. Since the conventional inverter is as large as a majority gate, several variations of the inverter have been developed as shown in 5.3. All the inverter variations in 5.3 are used for the implementations of this dissertation since their different relative positions of the input cells and the output cells are very useful for circuit optimization.

The majority gate in QCA is configured as shown in Figure 5.4 and is

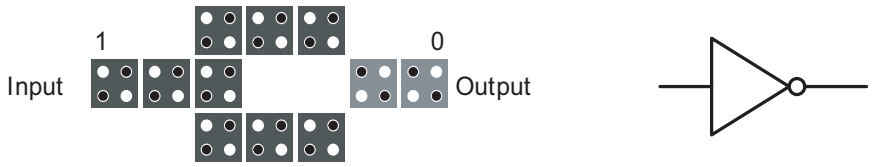


Figure 5.2: Layout and schematic symbol of a conventional inverter in QCA.

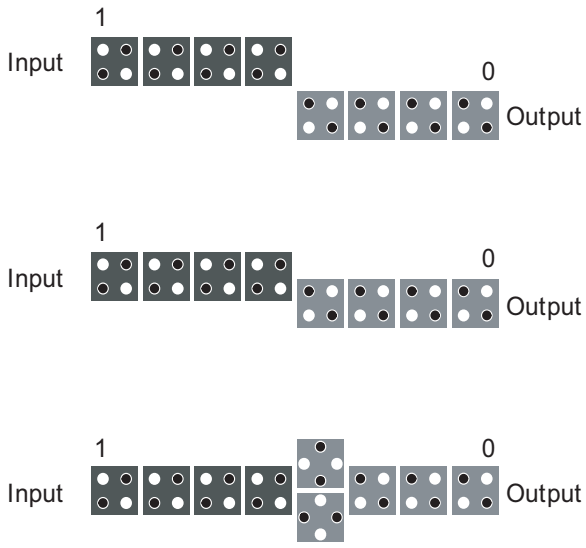


Figure 5.3: Layout of various inverters in QCA.

expressed using the following logic equation:

$$M(A, B, C) = A \cdot B + B \cdot C + C \cdot A$$

AND gates and OR gates are implemented by fixing one input of the majority gate as follows:

$$\begin{aligned} A \cdot B &= M(A, B, 0) \\ A + B &= M(A, B, 1) \end{aligned}$$

While the logic optimization methods for CMOS can be used for QCA circuits, majority logic reduction methods specialized for QCA [12, 44] make available further optimization especially for circuits using XOR gates. For example, a full adder circuit can be implemented using only three majority gates and several inverters.

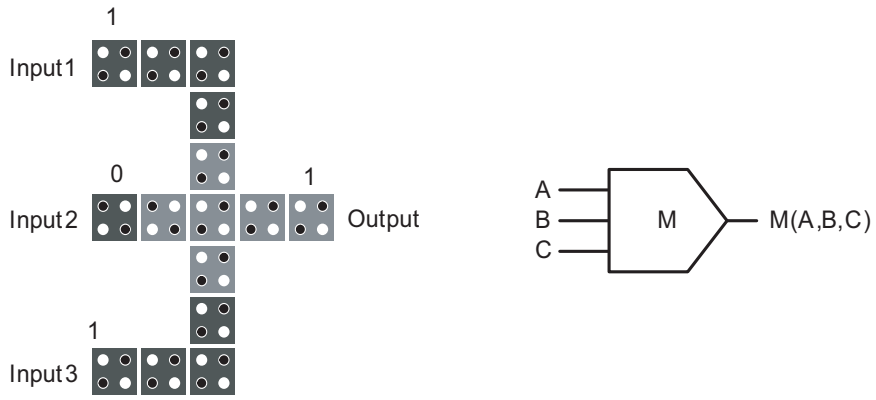


Figure 5.4: Layout and schematic symbol of a majority gate in QCA.

5.2.3 Clock Zones

All the QCA cells pertain to one of four clock zones, and the computations are performed sequentially in the same order as that of clock zones. Each clock zone has a different clock signal as shown in Figure 5.5. When the clock signal is high, the potential barriers between the quantum dots are low and the polarization is 0. When the clock signal is low, the electrons in a QCA cell are localized and the polarization will be held as ± 1. Using this 90° phase shifted signals, each clock zone has one of four phase states among Switch, Hold, Release, and Relax. A QCA cell begins computing during the Switch state and holds the polarization during the Hold state. The QCA cell prepares for the next computing during the Release state and the Relax state. QCA Wire transfers information using clock zones as shown in Figure 5.6.

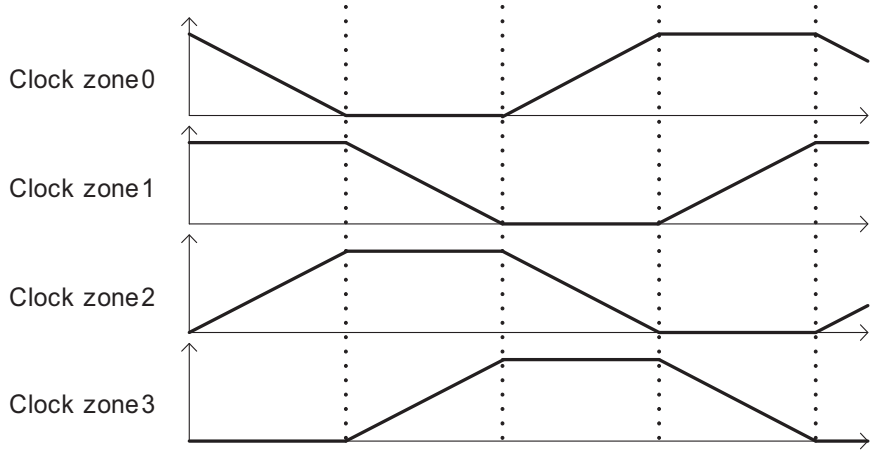


Figure 5.5: QCA clock signals for four clock zones.

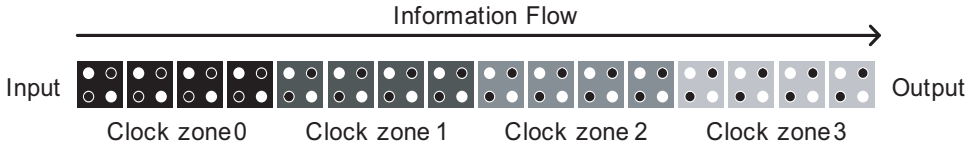


Figure 5.6: QCA wire with clock zones.

5.2.4 Coplanar Wire Crossing

There are two kinds of wire crossovers in QCA: coplanar wire crossovers and multi-layer crossovers. The coplanar wire crossovers [45] are implemented using regular cells and 45° rotated cells as shown in Figure 5.7. Signal A and Signal B can be transferred independently using only one layer. In contrast, the multi-layer crossovers require at least three layers for wire crossovers and via interconnections. Although the coplanar wire crossing has an advantage that

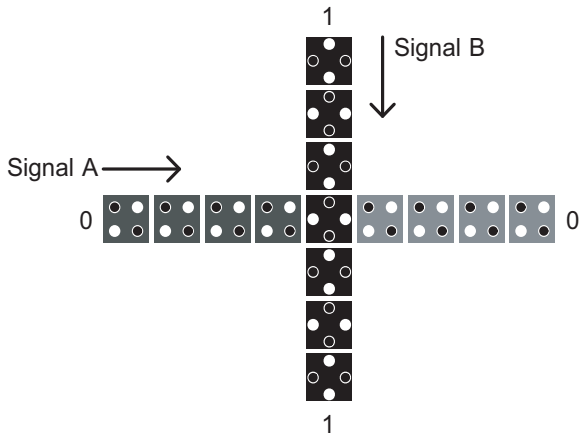


Figure 5.7: Layout of an example for coplanar wire crossovers.

it requires only one layer, they are susceptible to sneak noise from neighbor cells. To avoid problems due to the sneak noise, careful implementation based on design guidelines for robust operation is required. In conclusion, since the coplanar wire crossing seems more feasible for implementation, the divider in this dissertation is implemented using only the coplanar wire crossovers.

5.2.5 Conventional Goldschmidt Divider Architecture in QCA

A block diagram of a Goldschmidt divider for realization with CMOS technology is shown in Figure 5.8. It uses multiplexers and flip-flops that are controlled by a state machine during the iterations. This architecture poses a problem for QCA in synchronization due to the long delays between the state machine and the multiplexers. Also it does not take advantage of the inherent deep pipeline stages that are available in QCA. Thus a new architecture is required.

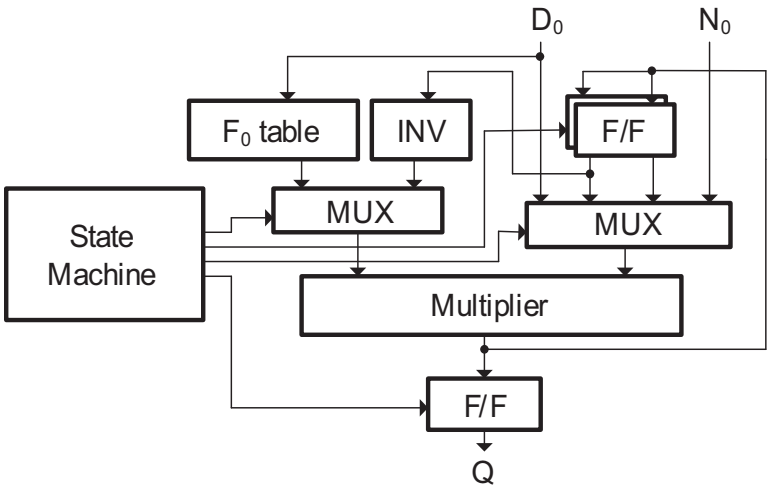


Figure 5.8: Goldschmidt divider block diagram for CMOS.

5.3 Data Tag Method for Iterative Computation

5.3.1 Problems in Using State Machines for QCA

Conventional iterative computation units using state machines as shown in Figure 5.9 are difficult to implement due to the long wire delays in QCA. Since wires are implemented by QCA cells like those used to construct gates, they have a delay that is similar to that of the gates. In addition, delays from a state machine to the units to be controlled vary according to the length of the wires. Due to this irregular wire delay, it is difficult to synchronize the inputs to units that are at a long distance from the state machine.

5.3.2 Data Tag Method

To resolve the problem of state machines for QCA, a data tag method is introduced as shown in Figure 5.10. In this method, data tags are associated

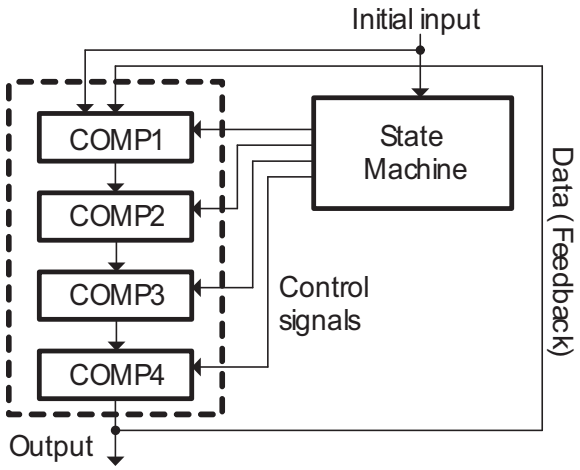


Figure 5.9: Computation unit implementations using a state machine.

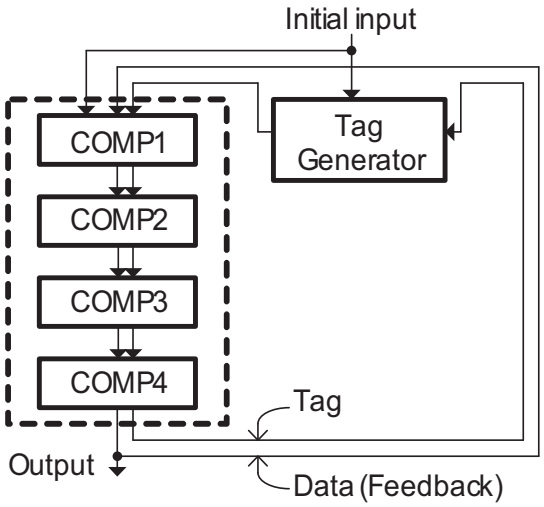


Figure 5.10: Computation unit implementations using the data tag method.

with the data, and local tag decoders generate control signals for the units (i.e., the computational circuits). The tags are transferred with the data through QCA pipeline stages, and they let the local tag decoders generate control signals appropriate to each datum. Since the tags travel together with the data and local tag decoders output appropriate control signals for the units, the synchronization issues that are a problem in state machines are eliminated. The data tag method is very efficient for QCA since flip-flops are generated inherently between gates and wires.

Another advantage of the data tag architecture is that each datum on a data path can be processed differently according to the tag information. For example, in typical Goldschmidt dividers for CMOS, a new division cannot be started until the previous division is completed. There are many pipeline stages in QCA, and most stages may be idle during iterations. With the data tag method, each datum on a data path can be processed by the operation that is required for that stage. Since divisions at different stages are processed in a time-skewed manner, a new division can be started while previous divisions are in progress if the initial pipeline stage of the data path is free. As a result, the throughput can be significantly increased.

5.3.3 Goldschmidt Divider with the Data Tag Method

A Goldschmidt divider has been designed using the data tag method as shown in Figure 5.11. To start a new division, the tag generator issues a new tag for the data. The local tag decoders control the multiplexers according

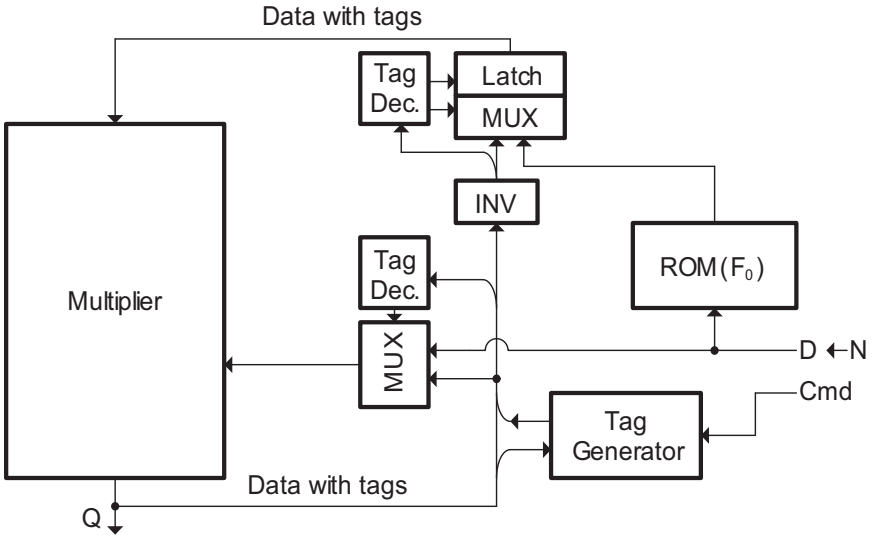


Figure 5.11: Block diagram of the Goldschmidt divider with the data tag method.

to the tags associated with the data. Each division is processed in its own iteration stage while other divisions are being performed. As a result, the throughput of the Goldschmidt divider is maximized. The throughput of the data tag method for a three iteration case is shown in Table 5.1.

Table 5.1: Throughput of the divider using the data tag method (1 iteration = N clocks, 2 data $(\mathrm{D}_0, \mathrm{N}_0)$ per division)

	Conventional method	Data tag method
Latency	$3N$	$3N$
Throughput	$1/(3N)$	$1/6$

5.4 Details of Implementation

5.4.1 Design Guidelines for Robust QCA Circuits

In order to design a robust circuit, the Goldschmidt divider has been designed using coplanar wire crossovers with the design guidelines suggested in [46, 47]. Coplanar wire crossovers are used for this dissertation since a physical implementation of multilayer crossovers has not been demonstrated yet. If multilayer crossovers are available for a design, the design can be implemented more efficiently. The design guidelines in [46] are kept except for a limitation on majority gate outputs. Robust operation of majority gates is attained by limiting the maximum cells that are driven by the output, which is verified using the coherence vector method. The maximum cell number for each circuit component in a clock zone is determined by simulations with sneak noise sources. For example, the maximum cell number for a simple wire is 14, and the minimum is 2.

5.4.2 Implementation of Goldschmidt Divider

The Goldschmidt iterative divider with the data tag method is implemented using a 12-bit array multiplier and a 3-bit ROM. D and N are input sequentially into the divider. The CMD signal is asserted together with D, and a new tag is generated from the tag generator. The tag decoders control the multiplexers and the latches using this tag. During the first iteration to normalize the denominator close to 1, the multiplexers are controlled in order that D and N are multiplied sequentially by F_0 from the reciprocal ROM. Af-

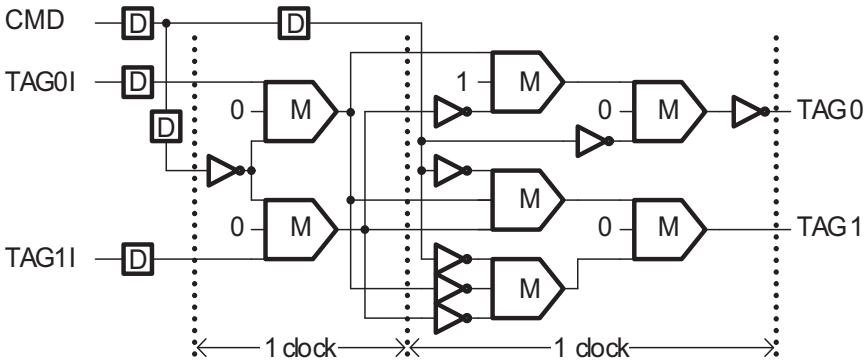
ter the first denominator normalization step is completed, the tag is changed for the next iteration through the tag generator. During the other iterations, the multiplexers select $\{D_i, N_i\}$ from the outputs of the multiplier and F_i that is computed using one's complement. After three iterations, the final quotient is computed, and the tag generator eliminates the tag.

5.4.2.1 Tag Generator

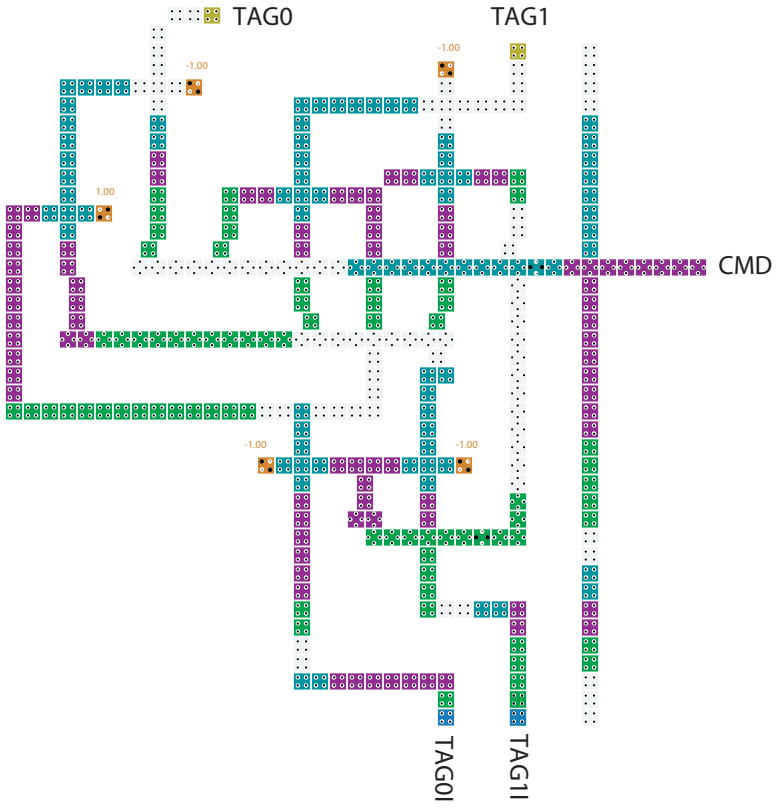
The tag generator creates a new tag and changes the tag. The tag generator has been implemented efficiently using majority logic reduction [44, 48] as shown in Figure 5.12. A new tag (TAG[1:0]=`01`) is generated when the CMD signal is asserted. In order to differentiate between D_i and N_i, the data tag is associated with only D_i, the first datum of a $\{D_i, N_i\}$ data set. N_i may be associated with a dummy data tag since QCA wires for the data tags cannot be reset during start-up. Therefore, the dummy data tag is eliminated by two AND gates as shown in Figure 5.12(a). On the other hand, if a tag arrives at the tag generator after an iteration step, the tag number is increased for the next iteration. After a division is completed, the data tag is eliminated (TAG[1:0]=`00`).

5.4.2.2 Tag Decoder and Multiplexers for $\{D_i, N_i\}$

The tag decoder and the multiplexers for $\{D_i, N_i\}$ are implemented as shown in Figure 5.13. When TAG[1:0] is `01`, the multiplexers select $\{D, N\}$ from the input data port. Since the multiplexers have to pass N one clock



(a)



(b)

Figure 5.12: Tag generator. (a) Schematic, (b) Layout.

after D is issued, the multiplexer selection signal in the tag decoder is held for two clocks as shown in Figure 5.13(a). Since the multiplexers for the less significant data have to be enabled earlier than other multiplexers for the pipelined operation of the multiplier, the tag decoder is located near the multiplexer for B[0], and the control signals for the multiplxers are in the reverse direction of the data flow.

5.4.2.3 Multiplexers for F_i

The multiplexers for F_i require latches in order to hold F_i for two clocks. During the two clocks, D_i and N_i are multiplied respectively by the value of F_i that was held by the latches. The latch is implemented by a SR latch using a majority gate [13] as shown in Figure 5.14(a). They are triggered when TAG[1:0] is not `00`.

5.4.2.4 $2^3 \times 3$-bit ROM Table

The $2^3 \times 3$-bit reciprocal ROM consists of a 3-bit decoder and an 8×3 ROM array as shown in Figure 5.15. All the ROM cells have the same access time, 7 clocks. The data are programmed by setting one input of the OR gate inside each ROM cell. Since the range of $D_i[0:11]$ is $0.5 \leq D < 1.0$ for the Goldschmidt division, $D_i[0:1]$ is always `01`, the input of the 3-bit ROM is $D_i[2:4]$. On the other hand, the output is $F_0[1:3]$ since $F_0[0]$ is always `1`.

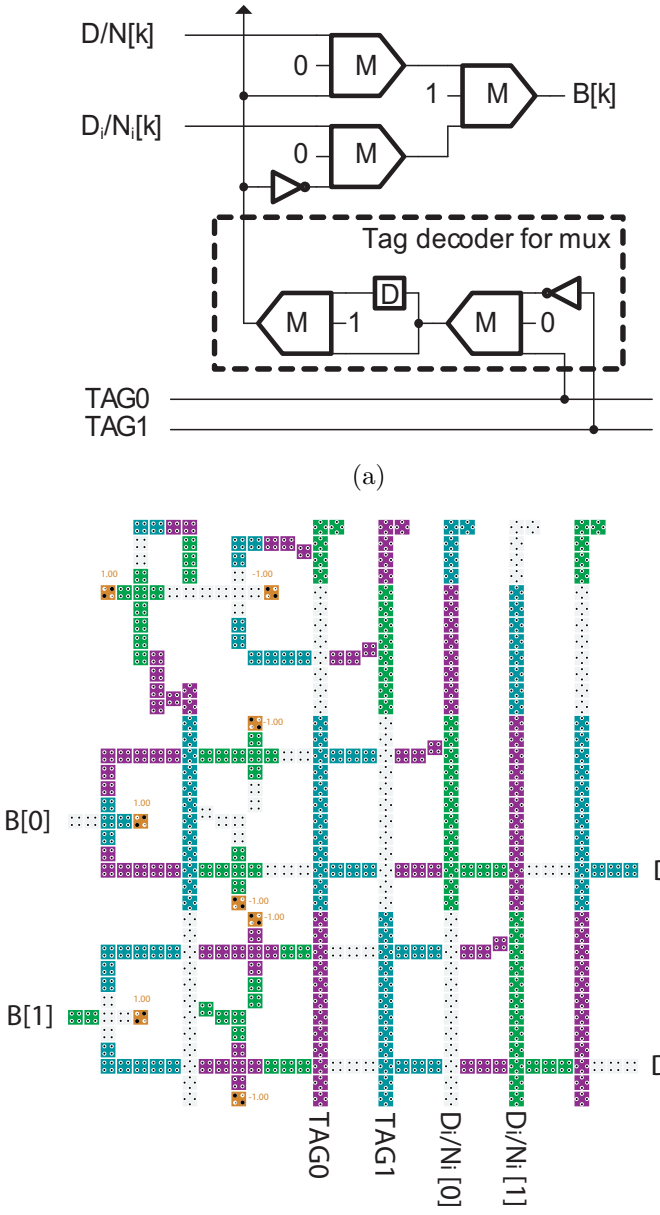
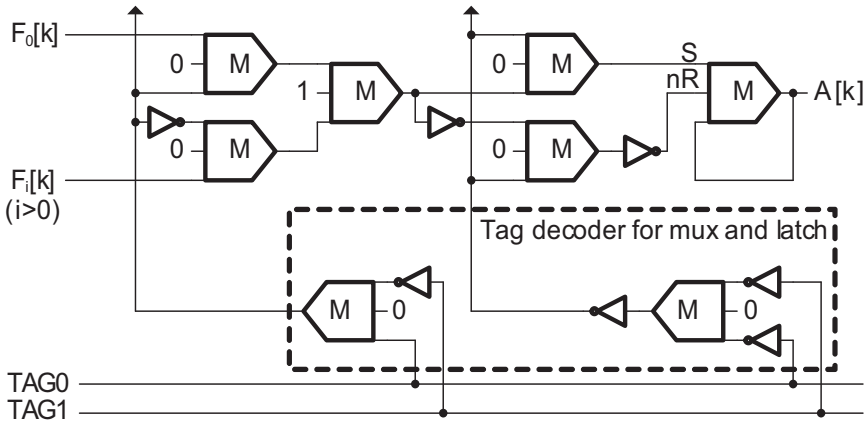
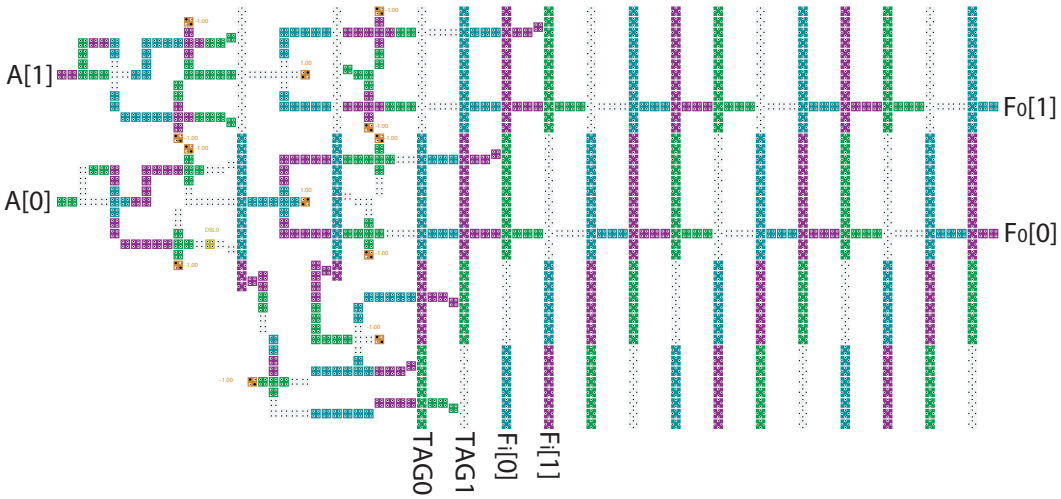


Figure 5.13: Tag decoder for multiplexers for $\{D_i, N_i\}$. (a) Schematic, (b) Layout for $k = 0, 1$.



(a)



(b)

Figure 5.14: Tag decoder for multiplexers and latches for F_i. (a) Schematic, (b) Layout for $k = 0, 1$.

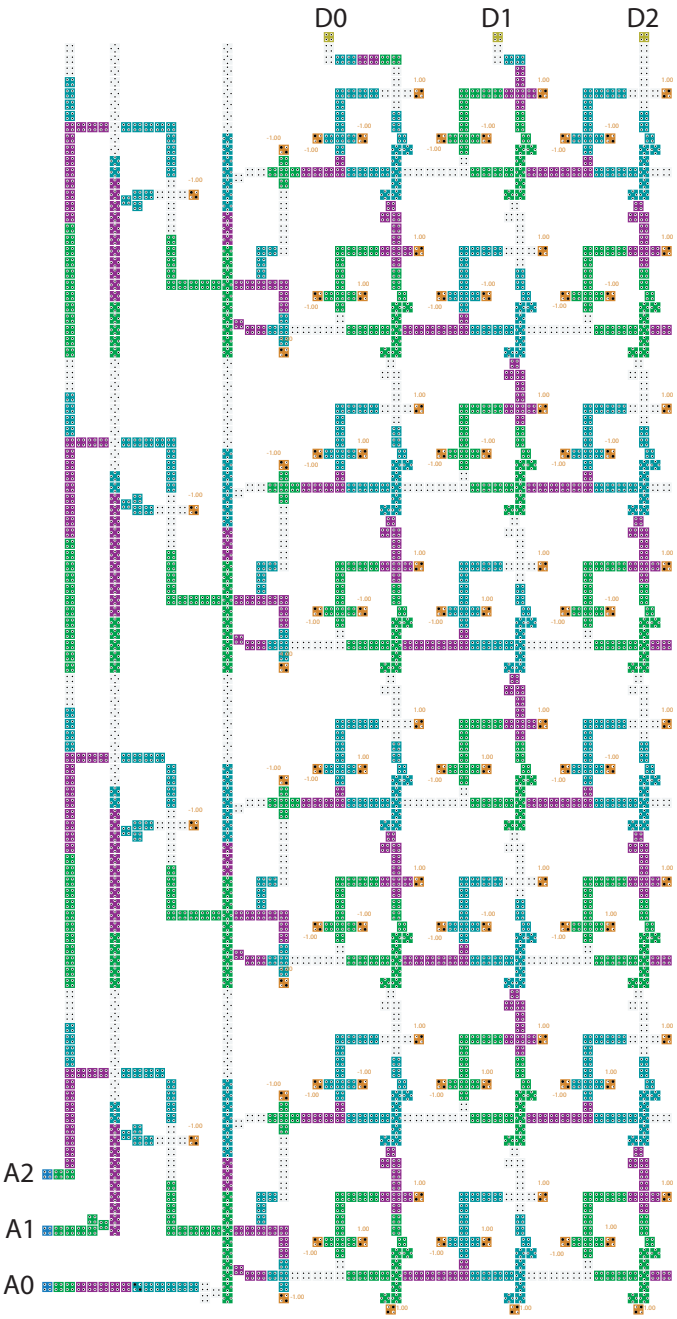


Figure 5.15: Layout of the 3-bit reciprocal ROM.

5.4.2.5 12-bit Array Multiplier

A 12-bit array multiplier is implemented for the Goldschmidt divider since array multipliers show the best performance in QCA [8, 49]. The multiplier cell is designed as shown in Figure 5.16 using a full adder with three majority gates [12, 44]. The schematic for 4 × 4-bit multiplier using the multiplier cell is shown in Figure 5.17. The cell layout is designed using coplanar crossovers, and it has signal delays of 1 clock for the carry output, 2 clocks for the sum, and an area of 20 × 29 cells. The 12-bit multiplier has two inputs, A[11:0] and B[11:0], and an output, M[11:0]. The unused SOUT signals of the

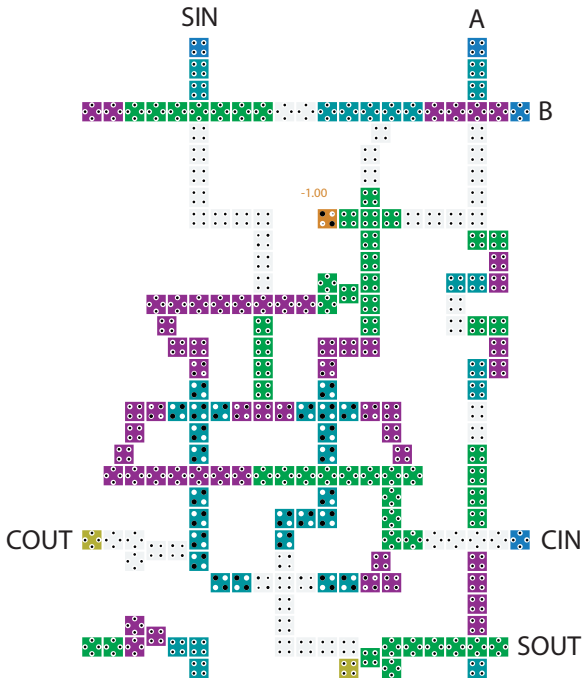


Figure 5.16: Layout of a cell for the array multiplier.

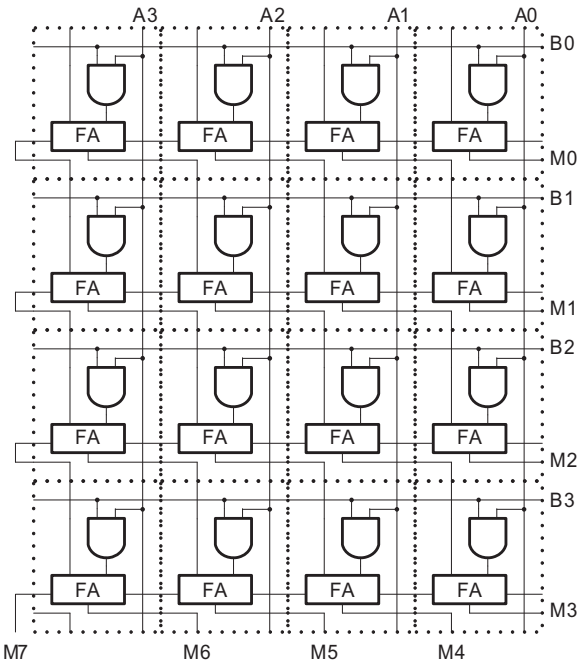


Figure 5.17: Schematic of a 4×4-bit multiplier using the multiplier cell.

rightmost vertical cells of the array multiplier should not be left unconnected since that violates the design guidelines in Section 5.4.1. Additional cells are attached to the unused SOUT outputs for robust transfers of the A[0] signal.

5.5 Simulation Results

The Goldschmidt divider has been implemented and simulated using QCADesigner v2.0.3 [50]. Most default parameters for bistable approximation in QCADesigner v2.0.3 are used except two parameters: the number of samples and the clock amplitude factor. Since the recommended number of samples is 1000 times the number of clocks in a test vector [50], the number of samples

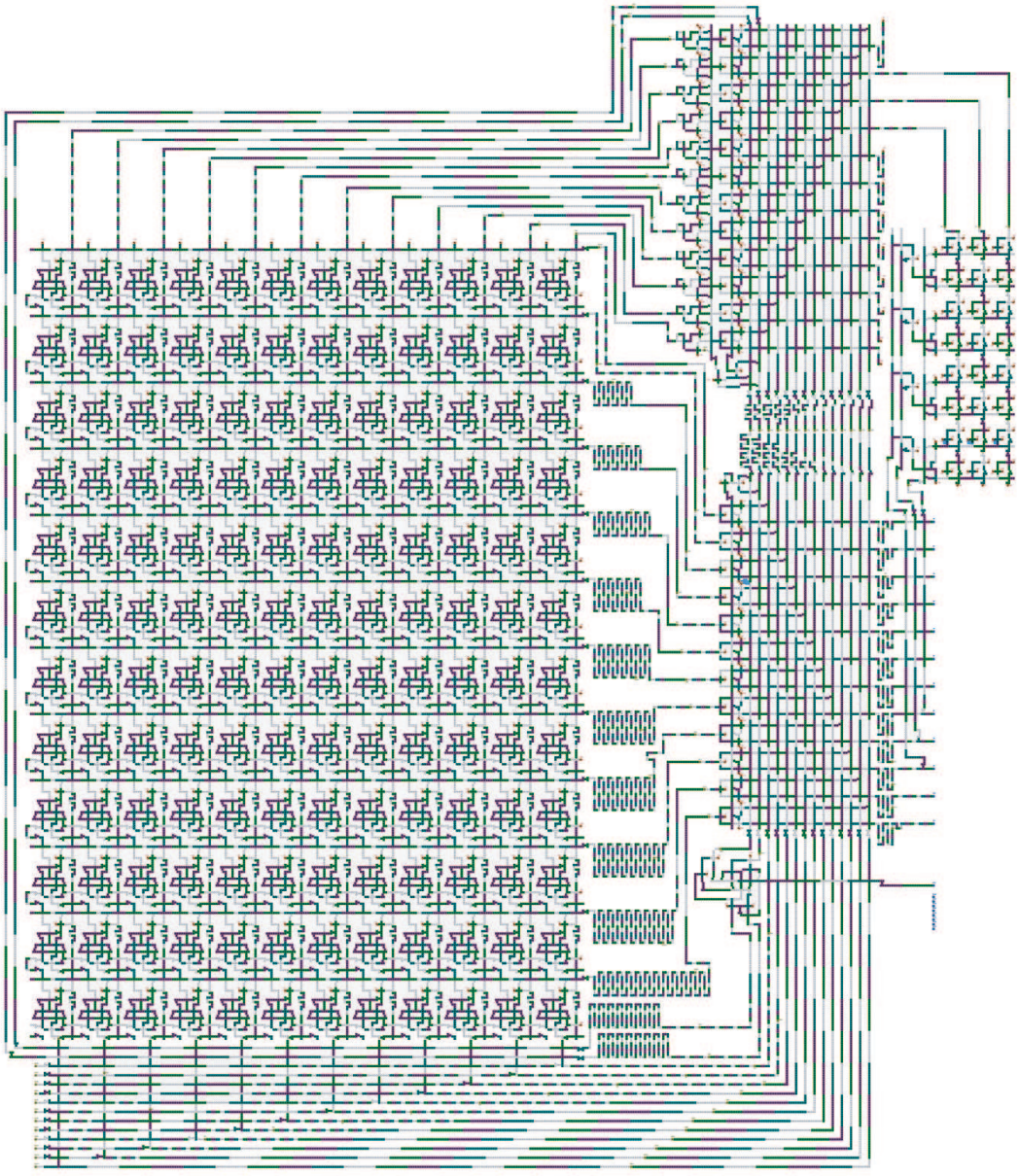
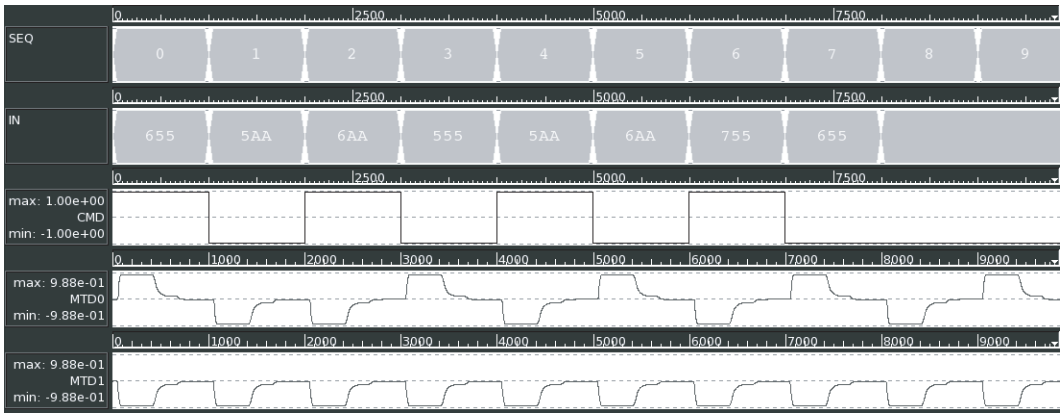
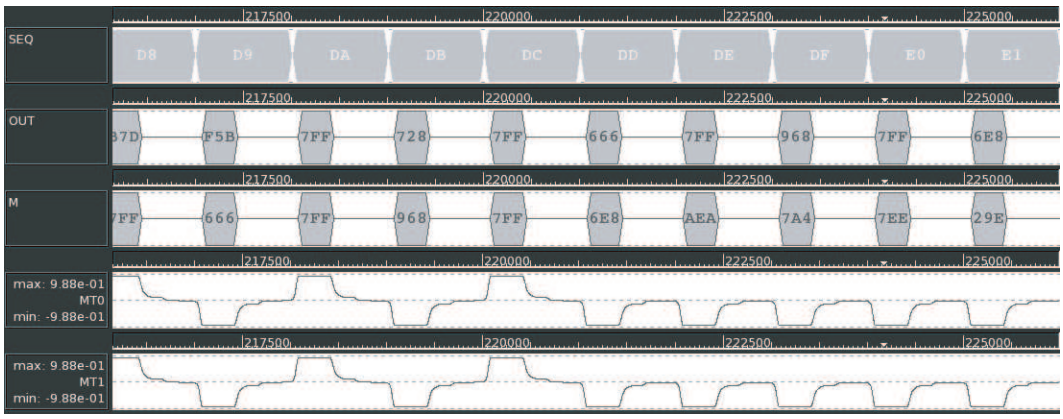


Figure 5.18: Layout of the Goldschmidt divider with a 12-bit multiplier.



(a)



(b)

Figure 5.19: Simulation results. (a) Input vectors for four consecutive divisions, (b) The output waveforms for the four quotients.

is determined to be 226000. Since adiabatic switching is effective to prevent a QCA system from relaxing to a wrong ground state [41], the clock amplitude factor is adjusted to 1.0 for more adiabatic switching. Other major parameters are as follows: size of QCA cell = $18nm \times 18nm$, center-to-center distance = $20nm$, radius of effect = $65nm$, and relative permittivity = 12.9.

Table 5.2: Delays of the functional units in the Goldschmidt divider

Functional unit	Delay (clock)
Tag generator	3
Multiplexer & Tag decoder	19
12-bit multiplier	46
Data bus for interconnects	5

The area for the Goldschmidt divider is $89.6\mu m^2$ ($8818nm \times 10158nm$), and the total number of the QCA cells is 55,562. The latency for a division is 219 clocks. The delays of the functional units are shown in Table 5.2.

The Goldschmidt divider is tested using bottom-up verification since a full simulation for a case takes about 7 hours. Each unit block is verified exhaustively, and then the full integration is tested. A full simulation result using a test vector for four consecutive divisions is shown in Figure 5.19. The input vectors are {`655h, 5aah, 6aah, 555h, 5aah, 6aah, 755h, 655h`}. In the waveforms, four correct quotients (N_2) start to come out from the 219-th clock, and four D_2s, which start from the 218-th clock, are shown correctly as `7ffh`.

5.6 Summary

A Goldschmidt divider (an iterative computational circuit) for quantum-dot cellular automata is implemented efficiently in a new architecture using data tags. The proposed data tag method avoids the synchronization prob-

lems that arise with conventional state machines in QCA due to the long delays between the state machines and the units to be controlled. In the proposed architecture, it is possible to issue a new division command at any iteration stage of a previous issued operation. Thus the throughput is significantly increased since multiple division computations can be performed in a time-skewed manner using one iterative divider. Using the data tag method, the fixed-point Goldschmidt divider using a 12-bit multiplier is implemented without synchronization problems.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This dissertation presents the algorithms and hardware designs for division by convergence. To reduce the required precision of the multiplier for Goldschmidt division, a rounding method with twice the error tolerance is proposed. To reduce the silicon area for the initial approximation table, a new approach that speeds the convergence compared to the standard quadratic convergence with a simple logic is suggested. To solve the difficulty in designing sequential circuits in QCA, a new architecture using a data tag method is proposed.

The proposed rounding method, which allows a larger error tolerance compared to the conventional rounding method, reduces the required precision of the multiplier for Goldschmidt division. Since the error tolerance of the new rounding method is twice that of the conventional method, iteration algorithms may have more error margin. It is implemented through a minor modification to the rounding constants of the multiplier. It also allows inclusive error bounds. The proposed method further reduces the required precision of the multiplier by considering the asymmetric error bounds of Goldschmidt

dividers where the factors are computed using a one's complement operation. In addition, it minimizes the maximum absolute error by shifting an asymmetric error span by a constant that is determined via an error analysis for a Goldschmidt divider. As a result, the proposed rounding method allows the multiplier of a 3-iteration Goldschmidt divider to be implemented using only three extra bits even though the factors are computed by one's complement operation. The proposed method has been verified for all four IEEE-754 rounding modes using a SystemC hardware model of the divider.

The required multiplier precision for Goldschmidt division is reduced using the DFQC method using that speeds the convergence compared to the standard quadratic convergence. Although division with cubic convergence has been regarded as impractical due to its complexity, the proposed method achieves nearly cubic convergence. It avoids the complex computations of true cubic convergence and achieves fast than quadratic convergence using the modified redundant binary Booth recoder and a simple approximate squaring computation. Due to the faster convergence, the size of the reciprocal ROM table can be reduced by 25% for the 2-iteration double precision floating-point Goldschmidt division. For 3-iteration double precision division, the table area can be reduced by 17%.

The new divider architecture for QCA using a data tag method shows that sequential circuits in QCA can be built efficiently without state machines. Since state machines for QCA often have synchronization problems due to the long delays between the state machines and the units to be controlled, the

proposed method may be a good solution for other iterative computations besides division. Since it is possible to issue a new division command at any iteration stage of a previous issued operation in the new division architecture, the throughput is significantly increased. Multiple division computations can be performed in a time skewed manner using the large number of inherent pipeline stages in QCA circuits. Although coplanar wire crossovers are very susceptible to noise, the design guidelines for robust QCA circuits produced a Goldschmidt divider circuit with more than 55000 QCA cells, which is simulated flawlessly during all the iteration cycles.

6.2 Published Results

This research has resulted in conference papers [31] and [51]. Journal papers [52] and [53] have been submitted. A revision was requested for [52] which has been submitted. Patents [54] and [55] have been disclosed and are expected to be filed.

6.3 Future Work

The proposed methods in this dissertation mitigate the two drawbacks of division by convergence. Future work should consider how to provide the exact remainder that is required to support IEEE-754 compliant rounding modes. Although the current correction step has the same complexity to the regular iteration, only the sign of the remainder is used for the final rounding. Since the output of the correction step is simple, it might be possible

to compute the correction without the complexity of a full iteration. As the conversion to normal binary in SRT division is avoided using an on-the-fly conversion and rounding methods [56], the correction stage for exact remainder may be efficiently implemented by a new method. If the correction step for the exact remainder were computed efficiently without a full iteration, the performance of division by convergence would be improved dramatically.

Although the data tag method provides an efficient architecture for Goldschmidt divider in QCA, the implemented divider can be further improved in the area and the latency. First, it can be modified to allow the use of a smaller serial parallel multiplier [7] since the current array multiplier accounts for about half of the area of the divider. Second, the latency of a division operation can be reduced since the early iteration steps do not require a full precision multiplication. If all the multiplications are computed using rectangular multipliers ($m \times n$ multipliers), the latency of each multiplication will be reduced, and the total latency is expected to be less than the current implementation. In this case, the new architecture should trade off between the increased amount of the area and the decreased amount of the latency. Also division algorithms other than division by convergence need to be analyzed in order to find the most efficient division architecture for QCA. Although division is a topic that has a long research history, there is still a lot of work to be done.

Bibliography

[1] S. F. Oberman and M. J. Flynn, "Design issues in division and other floating-point operations," *IEEE Transactions on Computers*, vol. 46, pp. 154–161, 1997.

[2] S. F. Oberman and M. J. Flynn, "Division algorithms and implementations," *IEEE Transactions on Computers*, vol. 46, pp. 833–854, 1997.

[3] S. F. Oberman, "Floating-point division and square root algorithms and implementation in the AMD-K7 microprocessor," *Proceedings of the 14th IEEE Symposium on Computer Arithmetic*, pp. 106–115, 1999.

[4] R. E. Goldschmidt, *Applications of Division by Convergence*, Master's thesis, Massachusetts Institute of Technology, 1964.

[5] C. S. Lent, P. D. Tougaw, W. Porod, and G. H. Bernstein, "Quantum cellular automata," *Nanotechnology*, vol. 4, pp. 49–57, 1993.

[6] C. S. Lent, P. D. Tougaw, and W. Porod, "Quantum cellular automata: the physics of computing with arrays of quantum dot molecules," *Proceedings of Workshop on Physics and Computation*, pp. 5–13, 1994.

[7] H. Cho and E. E. Swartzlander, Jr., "Adder and multiplier design in quantum-dot cellular automata," *IEEE Transactions on Computers*, vol. 58, pp. 721–727, 2009.

[8] S. Kim and E. E. Swartzlander, Jr., "Parallel multipliers for quantum-dot cellular automata," *IEEE Nanotechnology Materials and Devices Conference*, pp. 68–72, 2009.

[9] E. M. Schwarz, "Rounding for quadratically converging algorithms for division and square root," *Proceedings of the 29th Asilomar Conference on Signals, Systems and Computers*, pp. 600–603, 1995.

[10] S. F. Oberman and M. J. Flynn, *Fast IEEE Rounding for Division by Functional Iteration*, Stanford University, Tech. Rep. CSL-TR-96-700, Jul. 1996.

[11] M. J. Schulte, D. Tan, and C. E. Lemonds, "Floating-point division algorithms for an x86 microprocessor with a rectangular multiplier," in *25th International Conference on Computer Design*, pp. 304–310, 2007.

[12] W. Wang, K. Walus, and G. A. Jullien, "Quantum-dot cellular automata adders," *Third IEEE Conference on Nanotechnology*, vol. 1, pp. 461–464, 2003.

[13] J. Huang, M. Momenzadeh, and F. Lombardi, "Design of sequential circuits by quantum-dot cellular automata," *Microelectronics Journal*, vol. 38, pp. 525–537, 2007.

[14] H. Cho and E. E. Swartzlander, Jr., "Adder designs and analyses for quantum-dot cellular automata," *IEEE Transactions on Nanotechnology*, vol. 6, pp. 374–383, 2007.

[15] H. Cho and E. E. Swartzlander, Jr., "Serial parallel multiplier design in quantum-dot cellular automata," *Proceedings of the 18th IEEE Symposium on Computer Arithmetic*, pp. 7–15, 2007.

[16] D. DasSarma and D. W. Matula, "Measuring the accuracy of ROM reciprocal tables," *IEEE Transactions on Computers*, vol. 43, pp. 932–940, 1994.

[17] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. San Francisco, CA: Morgan Kaufmann Publishers, 2004.

[18] C. Iordache and D. W. Matula, "On infinitely precise rounding for division, square root, reciprocal and square root reciprocal," *Proceedings of the 14th IEEE Symposium on Computer Arithmetic*, pp. 233–240, 1999.

[19] P. W. Markstein, "Computation of elementary functions on the IBM RISC system/6000 processor," *IBM Journal of Research and Development*, vol. 34, pp. 111–119, 1990.

[20] H. M. Darley, M. C. Gill, D. C. Earl, D. T. Ngo, P. C. Wang, M. B. L. Hipona, and J. Dodrill, "Floating point/integer processor with divide and square root functions," U.S. Patent 4,878,190, Oct. 31, 1989.

[21] D. Piso and J. D. Bruguera, "A new rounding algorithm for variable latency division and square root implementations," *11th Euromicro Symposium on Digital Systems Design*, pp. 760–767, 2008.

[22] G. Even, P.-M. Seidel, and W. E. Ferguson, "A parametric error analysis of Goldschmidt's division algorithm," *Proceedings of the 16th IEEE Symposium on Computer Arithmetic*, pp. 165–171, 2003.

[23] D. Piso and J. D. Bruguera, "Optimizing the multiplier design for Goldschmidt's division and reciprocal units," *XXI Conference on Design of Circuits and Integrated Systems*, Barcelona, Spain, sec. 1C.3, 2006.

[24] D. DasSarma and D. W. Matula, "Faithful bipartite ROM reciprocal tables," *Proceedings of the 12th IEEE Symposium on Computer Arithmetic*, pp. 17–28, 1995.

[25] M. J. Schulte and J. E. Stine, "Symmetric bipartite tables for accurate function approximation," *Proceedings of the 13th IEEE Symposium on Computer Arithmetic*, pp. 175–183, 1997.

[26] M. J. Schulte, J. Omar, and E. E. Swartzlander, Jr., "Optimal initial approximation for the Newton-Raphson division algorithm," *Computing*, vol. 53, pp. 233–242, 1994.

[27] J.-A. Pineiro, S. F. Oberman, J.-M. Muller, and J. D. Bruguera, "High-speed function approximation using a minimax quadratic interpolator," *IEEE Transactions on Computers*, vol. 54, pp. 304–318, 2005.

[28] J.-A. Pineiro and J. D. Bruguera, "High-speed double-precision computation of reciprocal, division, square root, and inverse square root," *IEEE Transactions on Computers*, vol. 51, pp. 1377–1388, 2002.

[29] M. D. Ercegovac, L. Imbert, D. W. Matula, J. M. Muller, and G. Wei, "Improving Goldschmidt division, square root, and square root reciprocal," *IEEE Transactions on Computers*, vol. 49, pp. 759–763, 2000.

[30] M. Ito, N. Takagi, and S. Yajima, "Efficient initial approximation and fast converging methods for division and square root," *Proceedings of the 12th IEEE Symposium on Computer Arithmetic*, pp. 2–9, 1995.

[31] I. Kong and E. E. Swartzlander, Jr., "A rounding method with improved error tolerance for division by convergence," *Proceedings of the 42nd Asilomar Conference on Signals, Systems and Computers*, pp. 1814–1818, 2008.

[32] N. Quach, N. Takagi, and M. J. Flynn, *On Fast IEEE Rounding*, Stanford University, Tech. Rep. CSL-TR-91-459, 1991.

[33] J. Pineiro, J. Bruguera, and J. Muller, "Faithful powering computation using table look-up and a fused accumulation tree," *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, pp. 40–47, 2001.

[34] C. N. Lyu and D. W. Matula, "Redundant binary booth recoding," *Proceedings of the 12th IEEE Symposium on Computer Arithmetic*, pp. 50–57, 1995.

[35] G. Even and P.-M. Seidel, "Pipelined multiplicative division with IEEE rounding," *International Conference on Computer Design*, pp. 240–245, 2003.

[36] S. Haijun, S. Zhibiao, Z. Gang, and Z. Ning, "The research on optimization techniques of 32-bit floating-point RISC microprocessor," *Proceedings of 2005 IEEE International Workshop on VLSI Design and Video Technology*, pp. 63–66, 2005.

[37] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Transactions on Electronic Computers*, vol. EC-10, pp. 389–400, 1961.

[38] G. A. Ruiz and M. Granda, "Efficient implementation of 3x for radix-8 encoding," *Microelectronics Journal*, vol. 39, pp. 152 – 159, 2008.

[39] E. M. Schwarz, I. R. M. Averill, and L. J. Sigal, "A radix-8 CMOS S/390 multiplier," *Proceedings of the 13th IEEE Symposium on Computer Arithmetic*, pp. 2–9, 1997.

[40] K. Walus, M. Mazur, G. Schulhof, and G. A. Jullien, "Simple 4-bit processor based on quantum-dot cellular automata (QCA)," *16th International Conference on Application-Specific Systems, Architecture and Processors*, pp. 288–293, 2005.

[41] C. S. Lent and P. D. Tougaw, "A device architecture for computing with quantum dots," *Proceedings of the IEEE*, vol. 85, pp. 541–557, 1997.

[42] M. T. Niemier and P. M. Kogge, "Logic in wire: using quantum dots to implement a microprocessor," *The 6th IEEE International Conference on Electronics, Circuits and Systems*, pp. 1211–1215, 1999.

[43] M. T. Niemier and P. M. Kogge, "Problems in designing with QCAs: Layout = timing," *International Journal of Circuit Theory and Applications*, vol. 29, pp. 49–62, 2001.

[44] R. Zhang, K. Walus, W. Wang, and G. A. Jullien, "A method of majority logic reduction for quantum cellular automata," *IEEE Transactions on Nanotechnology*, vol. 3, pp. 443–450, 2004.

[45] D. Tougaw and C. S. Lent, "Logical devices implemented using quantum cellular automata," *Journal of Applied Physics*, vol. 75, pp. 1818–1825, 1994.

[46] K. Kim, K. Wu, and R. Karri, "Towards designing robust QCA architectures in the presence of sneak noise paths," *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1214–1219, 2005.

[47] K. Kim, K. Wu, and R. Karri, "The robust QCA adder designs using composable QCA building blocks," *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, vol. 26, pp. 176–183, 2007.

[48] K. Walus, G. Schulhof, G. A. Jullien, R. Zhang, and W. Wang, "Circuit design based on majority gates for applications with quantum-dot cellular automata," *Proceedings of the 38th Asilomar Conference on Signals, Systems and Computers*, pp. 1354–1357, 2004.

[49] I. Hänninen and J. Takala, "Pipelined array multiplier based on quantum-dot cellular automata," *18th European Conference on Circuit Theory and*

Design, pp. 938–941, 2007.

[50] K. Walus, T. J. Dysart, G. A. Jullien, and R. A. Budiman, "QCADesigner: A rapid design and simulation tool for quantum-dot cellular automata," *IEEE Transactions on Nanotechnology*, vol. 3, pp. 26–31, 2004.

[51] I. Kong, E. E. Swartzlander, Jr., and S.-W. Kim, "Design of a Goldschmidt iterative divider for quantum-dot cellular automata," *IEEE/ACM International Symposium on Nanoscale Architectures*, pp. 47–50, 2009.

[52] I. Kong and E. E. Swartzlander, Jr., "A Goldschmidt division method with fast than quadratic convergence," *IEEE Transactions on Very Large Scale Integration Systems*, submitted May 21, 2009, revised Sep. 21, 2009.

[53] I. Kong and E. E. Swartzlander, Jr., "A rounding method to reduce the required multiplier precision for Goldschmidt division," *IEEE Transactions on Computers*, submitted Sep. 30, 2009.

[54] I. Kong and E. E. Swartzlander, Jr., "Divider with faster than quadratic convergence," Patent disclosure OTC-5628-SWA, May 22, 2009.

[55] I. Kong and E. E. Swartzlander, Jr., "Data tag control method for Quantum-dot Cellular Automata," Patent disclosure OTC-5691-SWA, Sep. 30, 2009.

[56] M. D. Ercegovac and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Norwell, MA: Kluwer Academic Publishers, 1994.

Vita

Inwook Kong received the Bachelor of Science degree in Electrical Engineering from Yonsei University, Seoul Korea, in 1995. He received the Master of Engineering in Electrical Engineering with a thesis on signal processing using Fuzzy clustering in 1997. He has worked in the system LSI division of Samsung Electronics Co., LTD., Korea, where he developed ARM based mobile SOCs from 1997 to 2005. He joined at the University of Texas in August, 2005, where he started his graduate studies for Ph.D. degree in the integrated circuit and systems track of Electrical and Computer Engineering. He joined the application specific processor group in December, 2005. His research interests are high-speed computer arithmetic algorithms, VLSI circuit designs, application specific signal processing, and arithmetic algorithms on Quantum-dot Cellular Automata.

Permanent address: 3607 Greystone Dr. #514
Austin, Texas 78731-2227

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.