Modified Non-restoring Division Algorithm with Improved Delay Profile and Error Correction

Kihwan Jun and Earl E. Swartzlander, Jr.
Department of Electrical and Computer Engineering
University of Texas at Austin
Austin, Texas 78712 USA

Abstract - **This paper focuses on improving the performance of non-restoring division by reducing the delay and finding a correct quotient quickly. Although the non-restoring division algorithm is the fastest and has less complexity than other radix-2 digit recurrent division algorithms, there are still some possibilities to enhance its performance. To improve its performance, two new approaches are proposed here. For the first proposed approach, a non-restoring divider with a modified algorithm is presented. The new algorithm changes the order of the flowchart, which reduces one unit delay of the multiplexer per iteration. Secondly, a new method to find a correct quotient is presented and it removes an error that the quotient is always odd number after a digit conversion from a digit converter from the quotient with digits 1 and -1 to a conventional binary number. The new logic to generate the LSB of the quotient quickly is also explained in this paper.**

I. Introduction

Computers have evolved rapidly since their creation. However, there is one thing that has not changed: The main purpose of computers is to do the arithmetic to run programs and applications. Basically, computers handle lots of numbers based on the three basic arithmetic operations of addition, multiplication and division. Compared to addition and multiplication, division is the least used operation. However, computers will experience performance degradation if division is ignored [1, 2, 3]. There are two kinds of division methods devised by researchers: digit recurrent division and division by convergence. Each method has its own advantages [1], however digit recurrence division is most common algorithm for division and square root in many floating point units, since it is simple and lower in complexity than division by convergence [2, 4, 5]. Restoring, non-restoring and SRT dividers are representative algorithms for digit recurrence division. This paper focuses on the total delay reduction of non-restoring division and efficient quotient error correction. To achieve these goals, an improved algorithm is presented in this paper.

II. Algorithm Modification for Reducing Delay

Although division by convergence has several advantages such as quadratic convergence for fast division, it has some disadvantages including a complex structure, a large area due to including a multiplier and a look-up table and these characteristics result in spending more power. So, digit recurrence division is useful if the system requires a simple structure and small area. The restoring non-performing division algorithm is the least complex of the four digit recurrence methods, but the non-restoring division algorithm is the fastest based on previous researches [6, 7]. However, there are a couple of ways to improve the performance of non-restoring division. First approach is rearranging the order of the computation. By doing this, the delay of the multiplexer for selecting the quotient digit and determining the partial remainder can be reduced by one unit delay per iteration. This reduction will rise as the number of bits are increased. This approach is similar to the way that a carry select adder selects its intermediate results based on its carry [8]

A. Order Rearrangement

Figure 1(a) shows the implementation of the standard non-restoring division algorithm for a 16-bit divider. When the numerator and denominator enter the divider first, the number of bits for both the numerator and denominator are extended from 16-bits to 17-bits to check their signs for either positive or negative. Then the complement logic changes the denominator to one's complement form in the very beginning stage. After this process, the determination block which consists of a multiplexer and an inverter checks the sign of the numerator (at the first stage) or current partial remainder (after the first stage) and sets the quotient digit and determine whether to use the denominator or its complement for calculating the next partial remainder along with the current partial remainder. Then the addition is performed to calculate the partial remainder for the next iteration. The worst case delay comes from the path starting at the determination block and ending at the carry lookahead adder.

The delay of a multiplexer can be reduced if the select signal for two input data reaches to the multiplexer before the two input data arrive. In the non-restoring division algorithm, the select signal is the most significant bit of the numerator or the current partial remainder. So, one unit of delay will be eliminated if the algorithm is modified to receive the numerator or the current partial remainder at the multiplexer before two input data arrive there. To achieve this goal, the order of the determination block and the carry lookahead adders is switched as shown in Figure 1(b). For the modified algorithm, the numerator or current partial remainder is at the determination block and the determination block selects the quotient digit before the adders calculates its possible partial remainders. So, the multiplexer delay can be reduced from three units of delay to two. Although this delay reduction is relatively small, n iterations means it can critically affect the overall delay profile. Doubling the number of the adders is a tradeoff for reducing the delay. A similar structure is used for a carry select adder [8, 9].

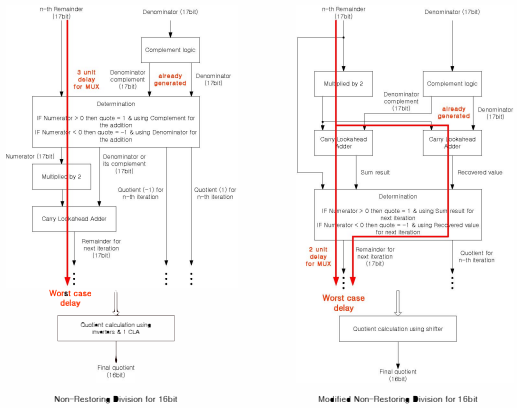


Fig. 1. (a) Standard non-restoring division algorithm for a 16-bit divider (b) Modified non-restoring division algorithm for a 16-bit divider

III. NOVEL METHOD TO CORRECT THE QUOTIENT ERROR

Non-restoring division has a quotient digit set of {1, -1} instead of the conventional binary digit set. Using the different quotient set reduces the delay of non-restoring division and it only requires one addition per iteration whereas the restoring division generally requires an average of 1.5 additions per iteration. However, there are a couple of drawbacks for using the different quotient set for the non-restoring division. First, the quotient with digits +1 and -1 must be converted to a conventional binary number using a digit conversion logic. On-the-fly conversion is commonly used and replaces the conventional digit converter as shown in Figure 2(a) with a 1-bit left shifter and a register as shown in Figure 2(b) [9,10]. There still has a problem regarding a least significant bit of a quotient for the non-restoring division algorithm although a digit converter generates the final quotient. The least significant bit of a quotient for the non-restoring division always sets to one regardless its next iteration since 2's complement of its -1 quotient word is used as a part of a conversion logic. It generates an error with a value of 2^{-n} for an n-bit divider. To eliminate this problem, a MSC generator to generate quickly the correct LSB of the quotient is explained in *Section B*. Second, one multiplexer for selecting the first quotient, q_0, can be removed since it does not affect the final quotient. So, the delay from one multiplexer will be eliminated as presented in *Section A*.

A. Algorithm Simplification

There is an additional process needed to convert the intermediate quotient with positive and negative digits into a conventional binary number. This process requires three steps. The first step is to separate the quotient word (with a mixture of both +1 and -1 bits) into two quotient words, P and N. P consists of only the +1 bits with zeroes in place of the -1 bits. N consists of +1 bits in place of the -1 bits and zeros in place of the +1 bits. In step 2, the two's complement of N is formed. Finally in step 3 the two's complement of N is added to P producing the positive quotient word. In this process, the leading bit can be ignored since the resulting number an unsigned binary number. The conversion algorithm includes forming P and N, forming a two's complement and an addition as shown in Figure 2.

The one's complement of N (the negative portion of a quotient) is exactly the same as P. The result is the same as if a 1-bit shift left of P is used instead of adding these two numbers together using a carry lookahead adder. A one is inserted at the LSB of the final quotient in all cases to convert the one's complement of the magnitude quotient bits to a two's complement. By adopting this scheme, the steps of computing the one's complement and the addition are removed from the non-restoring division algorithm and it allows the quotient to be calculated faster.

As discussed above, one multiplexer for selecting the first quotient, q_0, can be removed since it does not affect the final quotient. So, the delay from one multiplexer will be eliminated. Since the numerator entering into the floating-point divider is always positive [10], the leading bit can be ignored and one-bit quotient generated at the first stage is not required for the final quotient as shown in Figure 2, only the partial remainder is needed for the second stage. Therefore, the determination block that consists of one multiplexer at the first stage can be eliminated for algorithm simplification. It will also increase the calculation speed.

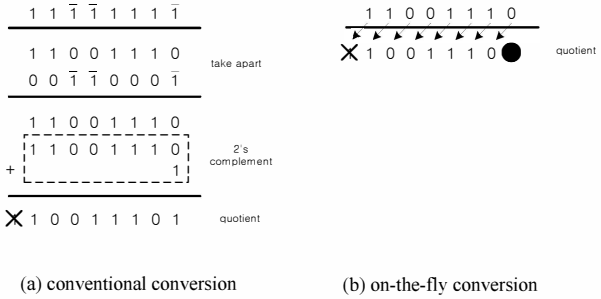


Fig. 2. Two method for converting from the set of digit +1 and -1 to a conventional binary number

B. The MSC generator

The least significant bit of a quotient for the non-restoring division is always set to one regardless its next iteration as shown in Figure 3 since 2's complement of its -1 quotient word is used as a part of a conversion logic. It generates an error of 2^{-n} for an n-bit divider. To remove the error, new logic to generate the correct LSB of the quotient quickly is explained in this section.

To enhance the accuracy for the least significant bit of the final quotient, an extra iteration is needed and it contains two carry lookahead adders and one inverter [9]. The total area and the worst case delay resulting from one iteration is increased considerably. The whole iteration is not required if the most significant bit of the last partial remainder is known. Since the least significant bit of the final quotient is directly obtained from the most significant bit of the last partial remainder, the exact value of the partial remainder is not necessary. If there is a faster and simpler alternative to find whether the partial remainder is positive or negative, then the extra iteration is no longer required.

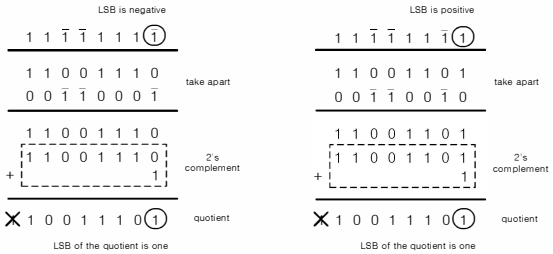
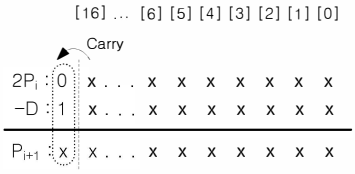
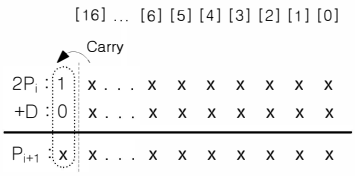


Fig. 3. The least significant bit for each final quotient in both cases

Figure 4 shows how to check the most significant bit of the partial remainder. Initially, both the numerator and the denominator are all positive since they come from the significands in the floating-point numbers and the significands may be divided by two, so that it fits to perform division as either a numerator or a denominator [11]. Based on the non-restoring algorithm, the positive current remainder is always paired with the negative denominator, the ones complement of the denominator, to calculate the next partial remainder. On the other hands, the negative current remainder is always paired with the positive denominator to calculate the next partial remainder as shown in Figure 4. In other words, the most significant bit of the denominator, +D, is always zero if the most significant bit of the doubled partial remainder, $2P_i$, is one and the most significant bit of the two's complement of the denominator, -D, is always one if the most significant bit of the doubled partial remainder, $2P_i$, is zero. Since the most significant bit of the next partial remainder, P_{i+1}, decides whether it is greater than zero or less than zero, the carry resulting from the addition at the n+1-th bit is one, then the most significant bit of the next partial remainder, P_{i+1}, is set to zero and the final quotient is generated as one. If the carry is not propagated to the most significant bit, the partial remainder is assumed to be less than zero and the last quotient bit is a zero.



(a) Partial remainder is positive



(b) Partial remainder is negative

Fig. 4. A new method to check the polarity of the partial remainder

A multilevel reverse most-significant-carry computation algorithm is used to calculate the most significant carry (MSC) quickly [12, 13]. It was originally devised for compound adders to compute an addition either with or without its carry, so early determination of the carry reduces the overall delay. The basic concept of MSC determination is as follows. First, check the carry propagation chain from MSB of both operands by using exclusive-OR gates. Then, check the MSC by using AND gates of both operand where the carry propagation chain is lost. Figure 5 shows an example of determining the MSC.

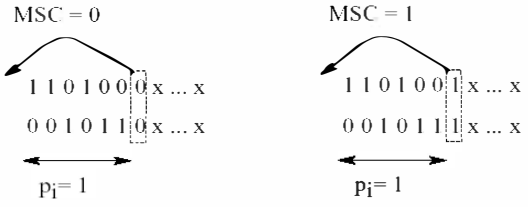


Fig. 5. Example of MSC determination

This can be done using both AND and OR gates without Exclusive-OR gates. The algorithm for the most significant carry (MSC) generator modified for non-restoring division is shown in Figure 6.

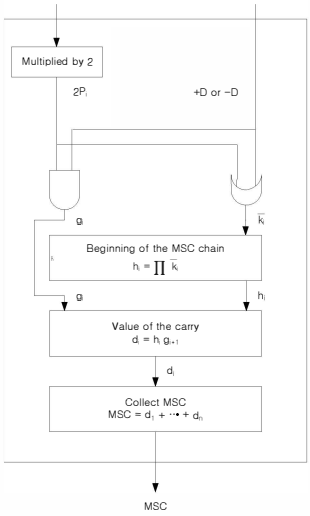


Fig. 6. A Modified MSC determination logic for the non-restoring division

The intermediate results g_i and k_i are generated by an AND operation and an OR operation, respectively on both operands. Then, h_i, the MSC chain, is generated by an AND operation from the most significant bit to each of the remaining bits in order. Using the previously generated g_{i+1} and h_i makes d_i successively. Finally, OR operations from the most significant bit to each of the remaining bits in order can generate the MSC. For verification purposes, the estimated delays and complexities for each error correction method are presented in Tables I and II.

Note that the one iteration method includes two carry lookahead adders, a multiplexer and an inverter. The error correction method with the MSC generator uses a multiplexer and a MSC generator. A 2-input AND or OR gate or an inverter is counted as 1 gate and has a delay of 1Δ.

TABLE I
WORST CASE DELAY COMPARISON BETWEEN ERROR CORRECTION METHODS

Worst case delay	One iternation	Proposed method	Difference
8 bit	15Δ	11Δ	4Δ (-26.7%)
16 bit	17Δ	12Δ	5Δ (-29.4%)
32 bit	21Δ	13Δ	8Δ (-38.1%)

TABLE II
COMPLEXITY COMPARISON BETWEEN ERROR CORRECTION METHODS

Complexity	One iternation	Proposed method	Difference
8 bit	277	71	224 (-74.4%)
16 bit	551	137	414 (-75.1%)
32 bit	1085	272	879 (-74.9%)

Table I shows that the worst case delays for the method with the MSC generator is almost 38% less than its delay per iteration for 32-bit division algorithm and it is evident that the MSC calculation algorithm is much faster than the one iteration method. Table II shows that the number of the logic gates used for the proposed method is much less than that of the conventional method.

A new algorithm for correcting the least significant bit of a quotient for the non-restoring division has been presented. The modified algorithm increases its speed and reduces the complexity by using a MSC generator. It has been analyzed that the proposed method to find the last quotient digit reduces the total delay by almost 38% per iteration. The reduction grows as the word size increases. The total area is decreased by over 74%. The new algorithm will be verified for correctness using Verilog models.

IV. SIMULATION RESULTS

In this section, simulations are performed for verification purposes. Nine different logic circuits are implemented using the Verilog Hardware Description Language (Verilog HDL). The FreePDK45nm version 1.0 standard cell library is used for the delay, the area and the power consumption. First, 8, 16 and 32-bit dividers using the standard non-restoring division algorithm that corrects the error by one more iteration are implemented and simulated. Then, the modified non-restoring dividers using either the MSC generator or the carry lookahead adder for error correction with 8, 16 and 32-bits are designed and simulated. To find the delays, the Synopsys Verilog Compiler Simulator (VCS) program is used to run the simulation with various Verilog HDL files for the dividers. Design Vision is used for finding the area and the power consumption. A hundred random vectors are generated as both the numerators and the denominators for each simulation and the delays in Table III are the average values of a hundred simulations.

Based on Table III, the modified non-restoring division algorithm with the MSC generator is the fastest and dramatically reduces the total delays compared to the standard non-restoring division algorithm. The delay difference between the two modified algorithms is between 0.72 and 1.42ns. For the 8-bit case, the modified non-restoring division algorithm with the MSC generator has a delay of 16.66 nanoseconds, which is almost 21% less than the standard. For 16-bit and 32-bit dividers, the delays are reduced by 21.2% and 21.3% respectively. As shown in Figure 6, the reduction in the delays becomes slightly larger as the number of bits is increased.

TABLE III
AVERAGE DELAYS FOR EACH DIVISION CIRCUIT

Delay	Standard NRD (a)	Modified NRD with iterations (b)	Modified NRD with iterations + MSC (c)	Diff. (a-c)
8 bit	21.04 ns	17.38 ns	16.66 ns	4.38 ns
16 bit	51.17 ns	41.30 ns	40.30 ns	10.87 ns
32 bit	112.22 ns	89.71 ns	88.29 ns	23.93 ns

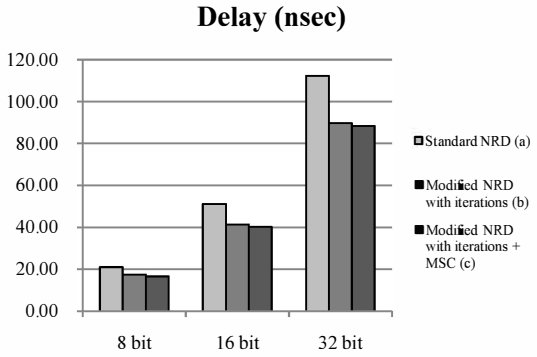


Fig. 6.Graph of the delays for each division circuit

Since the modified non-restoring division algorithm has almost twice as many adders as the standard non-restoring divider, it occupies more area than the standard divider even though some logic has been removed. The 8-bit non-restoring divider has an area of 2590μm^2 whereas the modified one with the MSC generator occupies 3999μm^2 which is almost a 54% increase as shown in Table IV. For 16-bit and 32-bit dividers, the areas are increased by 70.6% and 78.7%, respectively. This is the tradeoff between speed and area. In addition, just removing two carry lookahead adders does not have much of an effect on the total area since there are so many adders in a 32-bit modified non-restoring divider. By using the modified non-restoring division algorithm with the MSC generator, the area is reduced by 461μm^2 from that of the modified non-restoring division algorithm with one iteration for the 8-bit divider and the former one is almost 10% smaller than the latter one. For 16-bit and 32-bit dividers, the total areas are reduced by 5% and 2.5%, respectively.

TABLE IV
AREAS FOR EACH DIVISION CIRCUIT

Area	Standard NRD (a)	Modified NRD with iterations (b)	Modified NRD with iterations + MSC (c)	Diff. (a-c)
8 bit	2590μm^2	4460μm^2	3999μm^2	1409 μm^2
16 bit	10290μm^2	18486μm^2	17556μm^2	7266 μm^2
32 bit	40806μm^2	74793μm^2	72945μm^2	32139 μm^2

IV. CONCLUSION

A modified algorithm for non-restoring division has been presented. The modified algorithm increases its speed and enhances its precision as a result of algorithm modifications.

It has been verified that the modified non-restoring division reduces the total delay by almost 21% compared with the standard non-restoring division. The reduction increases as the word size increases. And, the least significant bit of the quotient is correctly generated via the modified non-restoring division algorithm. The total area is increased by over 70% as a tradeoff. This modified non-restoring division algorithm has been verified for a 45nm technology using Verilog models and simulations using Synopsys Verilog Compiler Simulator and Design Vision.

ACKNOWLEDGMENT

Earl Swartzlander's effort is supported in part by a grant from AMD.

REFERENCES

[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] Inwook Kong, "Modified Improved Algorithms and Hardware Designs for Division by Convergence," *Doctoral dissertation, University of Texas at Austin*, 2009.

[3] Peter Soderquist and Miriam Leeser, "Division and square root: choosing the right implementation," *IEEE Micro*, vol. 17, no. 4, pp. 56-66, July-Aug. 1997.

[4] Milos D. Ercegovac and Tomas Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*, Boston: Kluwer Academic Publishers, 1994.

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

[6] A.W. Burks, H.H. Goldstein, and J. von Neumann, *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument*, 2nd edition, Section 5.14, Princeton: Institute for Advanced Study, 1947.

[7] Gustavo Sutter, Jean-Pierre Deschamps, Gery Bioul and Eduardo Boemo, "Power Aware Dividers in FPGA," *Power and Timing Modeling, Optimization and Simulation 2004*, LNCS 3254, pp. 574–584, 2004.

[8] O. J. Bedrij, "Carry-select adder," *IRE Transactions on Electronic Computers*, vol. EC-11, pp. 340 –346, 1962.

[9] Behrooz Parhami, *Computer Arithmetic - Algorithms and Hardware Designs*, Oxford: Oxford University Press, 2000.

[10] ANSI/IEEE Standard 754-1985, *IEEE Standard for Binary Floating-Point Arithmetic*, 1985.

[11] Shlomo Waser and Michael J. Flynn, Introduction to Arithmetic for Digital Systems Designers, Holt, Rinehart & Winston, CBS College Publishing, 1982.

[12] J. D. Bruguera and Tomas Lang, "Multilevel Reverse Most-significant Carry Computation," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.9, no.6, pp.959-962, Dec. 2001.

[13] J. D. Bruguera and Tomas Lang, "Multilevel Reverse Carry Adder," *Proceedings of the 2000 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, pp. 155-162, 2000.