Fast Low-Power Shared Division and Square-Root Architecture*

Martin Kuhlmann, Keshab K. Parhi
Dept. of Electrical and Computer Engineering
University of Minnesota
200 Union Street SE, Minneapolis MN 55455, USA
Email: {kuhlmann, parhi}@ece.umn.edu

Abstract

This paper addresses a fast low-power implementation of a shared division and square-root architecture. Two approaches are considered in this paper; these include the SRT (Sweeney, Robertson and Tocher) approach which does not require prescaling and the GST (generalized Svoboda and Tung) approach which requires prescaling of the operands. This paper makes two important contributions. Although SRT division and square-root approaches and GST division approach have been known for long time, square-root architectures based on the GST approach have not been proposed so far. This paper, for the first time, develops a GST square-root architecture without requiring an additional division by the scaling factor after the square-root operation. Although various divider and square-root architectures have been compared with respect to speed, no tradeoffs with respect to power consumption of these architectures have been studied so far. Quantitative comparison of speed and power consumption of GST and SRT division/square-root units is the second main contribution of the paper. Shared divider and square-root units are designed based on the SRT and the GST approaches, in both minimally and maximally redundant radix-4 representations. Simulations demonstrate that the worst-case overall latency of the minimally-redundant GST architecture is 35% smaller compared to the SRT. Alternatively, for a fixed latency, the minimally-redundant GST architecture based division and square-root operations consume 42% and 20% less power, respectively, compared to the maximally-redundant SRT approach.

Keywords: Division, Square-root, Computer Arithmetic, Redundant Signed Digit, Radix-4, SRT, GST

*This work was supported by the Defense Advanced Research Projects Agency under contract number DA/DABT63-96-C-0050.

1 Introduction

Division and square-root are more challenging arithmetic operations than addition and multiplication. Division can be mathematically defined by the following expression:

$$X = Q \cdot Y + R, \tag{1}$$

where X, Y, Q, and R, respectively, represent the dividend, divisor, quotient and the remainder. If X and Y are represented in floating point and lie in the range $[1, 2)$ (according to IEEE-754 standard), then the quotient lies in the range $[1/2, 2)$. This result can be normalized by shifting the quotient by one bit to the left whenever it is in the range $[1/2, 1)$ and adjusting the exponent accordingly. In contrary to the division, square-root operation has only one input operand. The computation

$$X = Q \cdot Q + R, \tag{2}$$

computes the square-root quotient Q and the remainder R for the input X. Typically, the remainder R is of no interest. In IEEE-754 standard, the input operand is in the range $[0.25, 1)$ and the root quotient is in the range $[0.5, 1)$.

There are two main classes of division algorithms, the Multiplicative Algorithms (MA) and the Iterative Digit Recurrence Algorithms (IDRA). MA is based on multiplying the numerator and the denominator by constants, so that after each iteration the denominator converges closer to one while the numerator approaches the quotient, i.e.,

$$Q = \frac{X}{Y} = \frac{X \cdot R_0 \cdot R_1 \cdots R_{m-1}}{Y \cdot R_0 \cdot R_1 \cdots R_{m-1}} \rightarrow \frac{Q}{1}. \tag{3}$$

The number of iterations is proportional to log_2(word-length). However, since every iteration doesn't consist of simple addition/subtraction operations, but on two multiplications and one addition, very fast multipliers are needed to obtain the same efficiency as the IDRA. The large area requirement of this approach is a major drawback.

Iterative digit-recurrence division algorithms obtain the quotient one digit at a time (one quotient digit per clock cycle), most significant digit first. Thus, the number of iterations is directly proportional to the word-length of the dividend divided by half the radix. The IDRA is similar to the paper and pencil method [6], and is computed iteratively using

$$R_{i+1} = b \cdot R_i - q_i \cdot Y, \tag{4}$$

where R represents the remainder, b the radix, q the quotient digit and Y the divisor. The iterative algorithms can be implemented either using the generalized Svoboda-Tung (GST) approach [14, 17, 8, 9]) which requires prescaling of the operands or the SRT approach (Sweeney, Robertson and Tocher) [13, 16]) which does not require any prescaling. Prescaling of the operand does not alter the quotient, since

$$Q = \frac{X}{Y} = \frac{X \cdot k}{Y \cdot k} = \frac{X^{'}}{Y^{'}}. \tag{5}$$

This paper, for the first time, develops a GST square-root architecture without requiring an additional division by the scaling factor after the square-root operation as in [7].

This paper is organized as follows. Section 2 presents a brief overview of iterative digit recurrence algorithms. Section 3 presents the mathematical background for GST division and GST square-root operations. Section 4 presents the architecture of the GST divider/square-root unit and the estimated power consumption of the SRT and GST square-root architectures.

2 Iterative Digit Recurrence Algorithms

Both GST and SRT methods of iterative division and square-root are based on the non-restoring approach. Therefore, both approaches can take advantage of redundant number systems. Although radix-2 and radix-4 digit sets are most commonly used, much higher radix sets have also been considered. Besides the advantage of carrying out fast addition, the redundant digit sets have overlapping region, which results in an easy selection of the next quotient digit. This eliminates the need for a full length comparison of the remainder and the divisor. The decision criteria are chosen in such a way that only a few bits of the remainder and/or the divisor have to be checked to predict the next quotient digit. The SRT and the GST mainly differ with respect to the allowable range of the divisor. This can be easily explained using the P-D diagram shown in Fig. 1. The divisor is normalized according to the IEEE 754 standard to lie in the range $[1, 2)$. The SRT selects the next quotient digit according to the first few bits of the remainder and the divisor. In case of a minimally redundant radix-4 implementation, a total of 14 bits, 10 bits of the remainder and 4 bits of the divisor, have to be examined. The quotient digit selection

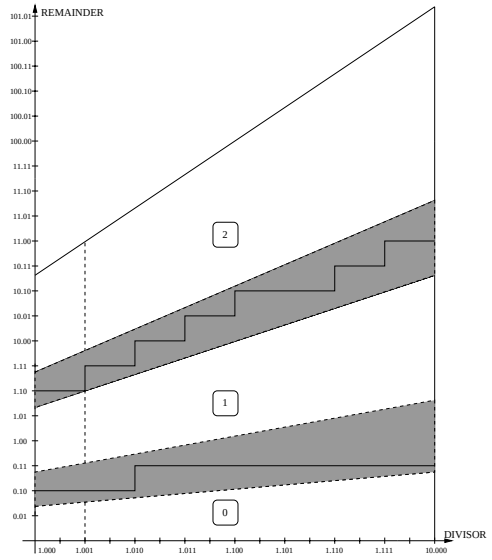

Figure 1. P-D diagram of a minimally redundant radix-4 digit set

is normally implemented as a PLA. The GST prescales the divisor and the dividend in such a way that the next quotient digit can be predicted without examining any bit of the divisor. The vertical dashed line in Fig. 1 shows this range. Due to this prescaling, only the first two digits of the remainder have to be checked to predict the next quotient digit. The two digits in radix-4 correspond to 6 bits. The drawback of the GST-division is caused by the additional clock cycles needed for prescaling. This drawback can be eliminated by realizing that the most significant quotient digit may be either one or two (those two possible digits cover the entire range $[0.5, 2)$). Hence, the most significant quotient digit can be predicted without examining the scaled dividend, at the expense of a few additional gates.

3 Mathematical Background for the GST Approach

3.1 Redundant Digit Representation

In the binary system, every word has only one distinct representation. A binary word is coded as:

$$w = a_0 \cdot 2^0 + a_1 \cdot 2^{-1} + a_2 \cdot 2^{-2} + a_3 \cdot 2^{-3} + a_4 \cdot 2^{-4} + a_5 \cdot 2^{-5} \ldots, \tag{6}$$

where a_i belongs to the set (0,1).

In the signed-digit representation $D < b, \alpha >$, the same word w is written as:

$$w = c_0 \cdot b^0 + c_1 \cdot b^{-1} + c_2 \cdot b^{-2} + c_3 \cdot b^{-3} + c_4 \cdot b^{-4} + c_5 \cdot b^{-5} \ldots, \tag{7}$$

where b represents the radix and c_i belongs to the set $(\overline{\alpha}, \overline{\alpha-1}..., 0, ..., \alpha-1, \alpha)$.

Therefore the decimal number 0.5, equivalent to binary 0.1, can be represented by either 0.1 or $1.\bar{1}$ in the radix-2 digit set. The main advantage of the redundant digit representation is the possible execution of fast, carry-propagation free, addition. Hence the critical path of an addition doesn't consist anymore on the word-length of the inputs but on the radix size. Thus a radix-2 adder has a critical path of one adder, while a minimally redundant radix-4 digit set has a critical path of two full-adders [2, 12].

However, the drawback of the redundant digit representation are the additional hardware costs. In the worst case, the consumed area is doubled compared to a binary implementation [15]. In the radix-2 case, there are three different values $(\bar{1}, 0, 1)$, so there is the need of having two bits for coding. Generally, the number of bits needed for coding a redundant digit set $D < b, \alpha >$ is equal to:

$$n = \lceil log_2(2 \cdot \alpha + 1) \rceil. \tag{8}$$

This leads to additional hardware wiring as well as an increase in consumed area.

3.2 Previous Work

Prescaling both operands does not alter the sought quotient, since:

$$Q = \frac{X}{Y} = \frac{X \cdot k}{Y \cdot k}. \tag{9}$$

3.2.1 The Arithmetic Condition

Every IDR algorithm has to fulfill certain arithmetic conditions, which limits the remainder into a certain range. Let us assume a symmetric redundant digit set $D < b, \alpha >$, where b represents the radix and α the maximum absolute value of the coefficients. Then, the maximum and minimum valid remainders are given by:

$$r^i_{max} = 0.\alpha\alpha\alpha...\alpha, \tag{10}$$

$$r^i_{min} = 0.\bar{\alpha}\bar{\alpha}...\bar{\alpha}. \tag{11}$$

The latter two equations can be rewritten as:

$$-\frac{\alpha}{b-1} < r^{i+1} < \frac{\alpha}{b-1}. \tag{12}$$

To obtain even an easier decision criteria, where the next quotient digit is equal to the most significant digit of the shifted remainder, the two MSDs have to be re-written in some cases. Assuming a minimally redundant radix-4 digit set, following cases have to be rewritten:

$$\begin{aligned} 2\bar{2} &\rightarrow 12, \\ 1\bar{2} &\rightarrow 02, \\ \bar{1}2 &\rightarrow 0\bar{2}, \\ \bar{2}2 &\rightarrow \bar{1}\bar{2}. \end{aligned}$$

The re-write block avoids a possible overflow and guarantees that the MSD of the new remainder is always zero. However, the re-write block only consists of a few gates, leading to a low latency and a low-power design.

3.2.2 Range of the divisor

The range of the divisor can be computed mathematically (besides the straightforward way by examining the P-D diagram). If the partial remainder $b \cdot R^i$ (= the next quotient bit q_i) is positive, then $b \cdot R^i$ is maximal when all the digits to the right of the decimal point have the greatest digit magnitude, i.e.,

$$b \cdot R^i_{max} = q_i.\alpha\alpha\alpha...\alpha. \tag{13}$$

Thus,

$$b \cdot R^i_{max} < q_i + \frac{\alpha}{b-1}. \tag{14}$$

The minimal R^i that could violate the arithmetic condition arises when R^i assumes the format (remember, the remainder $q_i.\bar{\alpha}\bar{\alpha}\bar{\alpha}...\bar{\alpha}$ is rewritten as $(q_i - 1).\alpha\bar{\alpha}\bar{\alpha}...\bar{\alpha}$, so the next quotient bit is no longer q_i).

$$b \cdot R^i_{min} = q_i.\bar{\beta}\bar{\alpha}\bar{\alpha}...\bar{\alpha}, \tag{15}$$

β being a positive integer such that $\beta < \alpha$. Hence,

$$b \cdot R^i_{min} > q_i - \frac{\beta}{b} - \frac{\alpha}{b \cdot (b-1)}. \tag{16}$$

By rewriting equation (4), we obtain:

$$r^i_1 \cdot Y = b \cdot R^i - R^{i+1}. \tag{17}$$

Replacing the values of $b \cdot R^i_{max}$ and $b \cdot R^i_{max}$ as well as R^{i+1}_{max} and R^{i+1}_{min}, we obtain,

$$-\frac{\alpha}{b-1} + q_i + \frac{\alpha}{b-1} \leq q_i \cdot Y < \frac{\alpha}{b-1} + q_i - \frac{\beta}{b} - \frac{\alpha}{b \cdot (b-1)}. \tag{18}$$

Simplifying this equation and dividing by q_i leads to:

$$1 \leq Y < 1 + \frac{\alpha - \beta}{b \cdot q_i}. \tag{19}$$

The larger q_i, the smaller the fraction on the right hand side. So q_i may be replaced by α, which leads to:

$$1 < Y < 1 + \frac{\alpha - \beta}{b \cdot \alpha}. \tag{20}$$

For a negative partial remainder, the same equations can be obtained. Assuming again a minimally redundant radix-4 digit set, $D < 4, 2 >$, the range of the divisor is limited between 1 and 9/8 [8].

3.3 The New Approach: GST Square-Root with Prescaling

Prescaling the operand X and the divisor Q does not alter the sought quotient, since:

$$Q = \frac{X}{Q} = \frac{X \cdot k}{Q \cdot k}. \tag{21}$$

Recall that the operand of the square-root is in the range of $[0.25, 1)$ and the result in $[0.5, 1)$. By multiplying the dividend by four and the divisor by two, we obtain a comparable equation to the division including the range of the divisor. The result of the square-root has only to be divided by two, which is equivalent to a simple shift operation. The square-root algorithm is quite similar to the iterative digit recurrence algorithm for division, and is given by

$$\begin{aligned} R_{i+1} &= b \cdot R_i - q_i \cdot (2 \cdot Q_{i-1} + b^{-i} \cdot q_i) \\ &= b \cdot R_i - q_i \cdot 2Q_i^1, \end{aligned} \tag{22}$$

where R represents the remainder, q_i the quotient digit, b the radix and Q the quotient computed up to the last iteration. In order to apply the GST to the square-root, Q_i^1 has to be scaled.

$$Q = \frac{X}{Q^1} = \frac{X \cdot k}{Q^1 \cdot k} = \frac{X^{'}}{Q^{1'}} \tag{23}$$

Since Q^1 changes every iteration, $Q^{1'}$ has to be updated accordingly (in this paper, scaled operands always have a dash). The result sought after every iteration is obtained by:

$$Q_i = Q_{i-1} + b^{-i} \cdot q_i. \tag{24}$$

Scaling equation (19) results in:

$$\begin{aligned} R_{i+1} &= b \cdot R_i - q_i \cdot (2 \cdot Q_{i-1} + b^{-i} \cdot q_i) \cdot k \\ &= b \cdot R_i - q_i \cdot 2Q_i^{1'}. \end{aligned} \tag{25}$$

The goal of this section is to obtain an equation of $Q_{i+1}^{1'}$ depending on $Q_i^{1'}$. Therefore, increasing i by 1 in the equation (22) results in:

$$\begin{aligned} R_{i+2} &= b \cdot R_{i+1} - q_{i+1} \cdot (2 \cdot Q_i + b^{-(i+1)} \cdot q_{i+1}) \cdot k \\ &= b \cdot R_{i+1} - q_{i+1} \cdot 2Q_{i+1}^{1'} \cdot k. \end{aligned} \tag{26}$$

Recalling that in equation (19)

$$2Q_i^1 = 2 \cdot Q_{i-1} + q_i \cdot b^{-i}, \tag{27}$$

and by comparing equations (19),(24) and (25), the sought equation is obtained as:

$$\begin{aligned} 2Q_{i+1}^{1'} &= (2 \cdot Q_i + q_{i+1} \cdot b^{-(i+1)}) \cdot k \\ &= (2 \cdot Q_{i-1} + 2q_i \cdot b^{-i} + q_{i+1} \cdot b^{-(i+1)}) \cdot k \\ &= 2 \cdot Q_i^{1'} + q_i \cdot b^{-i} \cdot k + q_{i+1} \cdot b^{-(i+1)} \cdot k. \end{aligned} \tag{28}$$

The next scaled divisor can now be obtained by just adding the quantities $q_i \cdot b^{-i} \cdot k$ and $q_{i+1} \cdot b^{-(i+1)} \cdot k$ to the previous scaled divisor. To avoid the possible overflow in the square-root operation, a similar rewrite condition is needed, which has just the opposite functionality of the rewrite block for the division.

3.3.1 Example

In this subsection, the effects of equation (28) are shown. Assuming an operand X=0.0101100110011_2 (=0.35_{10}) and therefore a scaling factor of k=0.111_2, X', the scaled operand, is obtained to X'=0.0100111001100110011_2. Writing X' as a minimally redundant radix-4 number, we get $X' = 0.11\bar{1}2121212$. In Fig. 2, the divisor $Q^{1'}$ is shown. Apparently, in every iteration, only the nine

iteration	q_i	$Q_i^{1'}$
i=0,	0.	0.0000
1	2	0.01110
2	2	0.1000110
3	$\bar{2}$	0.1000101010
4	$\bar{1}$	0.1000010000001
5	2	0.1000001001000010
6	$\bar{1}$	0.1000001001000000001
7	1	0.1000010010001011
8	$\bar{2}$	0.1000010010000011110
9	2	0.1000010010000010000110
10	2	0.1000010010000010001100110

Figure 2. The change of the scaled square-root divisor per iteration

least significant bits of the scaled square-root divisor may change. Hence, there is no need for a full length addition. Furthermore, the switching of these nine bits is caused by two additions. In the first addition of $q_i \cdot b^{-i} \cdot k$ at most seven bits may switch. In the second addition of $q_{i+1} \cdot b^{-(i+1)} \cdot k$, there is only the need of having a carry-ripple adder (CRA) of word-length five, because the last two significant bits are added to two zero bits. Thus, the scaling of the divisor Q' is relatively a low-cost operation.

4 Design Descriptions

4.1 Top Level Description of the GST

The top level description of the folded GST for Division and Square-Root is shown in Fig. 3. In Fig. 2, the top level description of the GST is presented. Every iteration is represented by a slice. The entire design only consists of combinational logic and there is no feedback. Every slice consists of a rewrite block, a decision criteria, multiplexers, hybrid-adder and an on-the-fly converter.

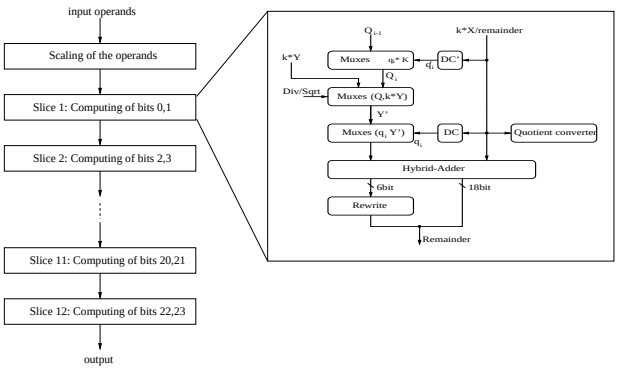

Figure 3. Top level description of the GST-division architecture

4.1.1 Prescaling

The first five MSBs of the divisor, Y, are used to select the constant K, by which the divisor and the dividend, X, are multiplied (Fig. 3). Since there are maximally three possible partial sums of the K-selector (Table 1), a Carry-Save-Adder (CSA) structure in parallel is used to reduce the number of partial sums to two. To obtain a fast prescaling operation, a multiplexer-adder [11] is employed to add the two partial sums. The structure of this adder is shown in Fig. 4. The multiplexer adder consists of a preprocessing unit,

Table 1. K-selection table

Range of Y			K	K in binary
1	$\leq Y \leq$	9/8	1	1/2 + 1/2
9/8	$\leq Y \leq$	19/16	0.9375	1/2 + 1/2 - 1/16
19/16	$\leq Y \leq$	5/4	0.875	1/2 + 1/4 + 1/8
5/4	$\leq Y \leq$	11/8	0.8125	1/2 + 1/4 + 1/16
11/8	$\leq Y \leq$	3/2	0.75	1/2 + 1/4
3/2	$\leq Y \leq$	13/8	0.6875	1/2 + 1/8 + 1/16
13/8	$\leq Y \leq$	57/32	0.625	1/2 + 1/8
57/32	$\leq Y \leq$	2	0.5625	1/2 + 1/16

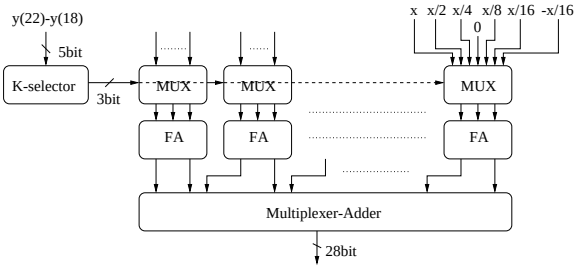

Figure 4. Structure for the prescaling block

where the input operands are rewritten for certain cases (if both input operands a_i and b_i are 1, then they are rewritten as zeros). In the post processing unit, the sum outputs are selected according to the carries generated in the fast carry generation block. The entire propagation delay of this multiplexer adder is equal to six multiplexer delays and one nand delay for the addition of two operands of the word-length of 28.

4.1.2 Iteration Block

The rewrite block guarantees that the most significant digit of the new remainder is always zero. The Decision Criteria (DC) (see Fig. 5) selects the new quotient digit according to the most significant digit of the output of the rewrite block. According to the selected quotient digit, the Muxes choose the multiple of the divisor Y, which is subtracted from the dividend. The structure of the Muxes and Hybrid-adder are presented in Fig. 6. The hybrid-adder consists of a full-adder and a plus-plus-minus adder, due to the coding of the minimally redundant radix-4 representation. Therefore, the critical path in every slice is equal to the delay of two full-adders, three multiplexers and a certain amount

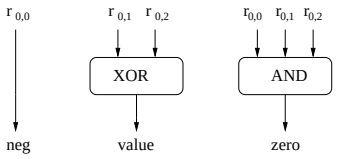

Figure 5. Random Logic for the Decision Criteria

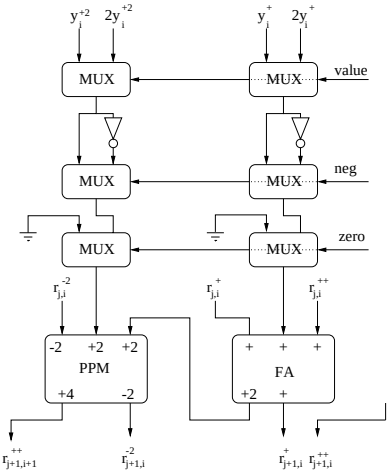

Figure 6. Possible multiplexer and adder structure for the GST

of small gates for decision criteria and rewrite block. The quotient converter is implemented using an on-the-fly converter. The structure of the converter is shown in Fig. 7. The multiplexers choose between the quotient and the quotient subtracted by one. $S^{\bar{1}}$ is selected if the next quotient digit is negative. The new two bits of the binary quotient are computed according to the equation:

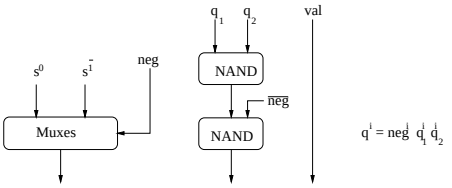

Figure 7. The structure of the on-the-fly converter

$$q_{bin} = neg \cdot b + q_i, \tag{29}$$

where b represents the radix and q_i the quotient digit. If q_i is smaller than 0, the signal *neg* is one, otherwise zero. The shift operation can be obtained by hardware wiring. The exponents of the divisor can be subtracted from the exponent of the dividend in a simple carry-ripple adder. However, since there is the possibility of obtaining a quotient in the range of $[1/2; 1)$, the quotient has to be normalized and a 1 has to be subtracted from the new exponent.

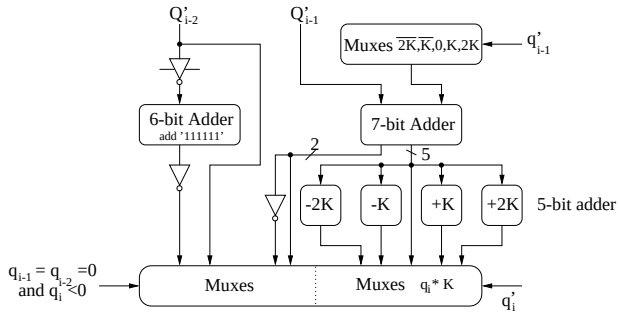

Figure 8. The structure of the scaled square-root quotient

5 Low Power Design

Since the delay time is proportional to $1/V_{dd}$, the supply voltage can be reduced to a certain value, so that the chosen frequency matches with the longest critical path. The propagation delay equation of a CMOS circuit is given by [3],

$$T_{delay} = \frac{C_{charge} V_{dd}}{k(V_{dd} - V_t)^2}. \tag{30}$$

When the propagation delay is less than the clock period by a factor M, we can reduce the supply voltage by a factor β such that T_{clk} is equal to T_{delay}. Hence,

$$\begin{aligned} T_{clk} &= M \cdot T_{delay}(V_{dd}) = T_{delay}(\beta \cdot V_{dd}) \\ &= \frac{C_{charge} \cdot \beta \cdot V_{dd}}{k \cdot (\beta \cdot V_{dd} - V_t)^2}. \end{aligned} \tag{31}$$

The last two equations can be simplified, so that the following result may be obtained:

$$M(\beta V_{dd} - V_t)^2 = \beta (V_{dd} - V_t)^2. \tag{32}$$

6 Simulation Results

All the necessary blocks have been designed using a 0.5 μm CMOS technology. To obtain a better comparison between the different architectures, all n-channel transistors have been implemented using the minimum width of 3λ, while the p-channel transistors have been implemented using a width of 6λ. Only drivers make use of a larger transistor width. The power estimations have been done by HSPICE.

All multiplexers and adders are implemented using transmission-gate based complementary pass transistor logic [1]. This has the effect of minimizing the critical path in the hybrid-adder and the multiplexers compared to a static CMOS implementation. The random logic used in the rewrite block and the decision criteria block is based on static CMOS design. To perform fast multiplication, a tree-based multiplexer adder (MXA) is utilized.

Simulation results have shown that the GST algorithm has the shortest critical path (the critical path is equal to the fastest possible clock period in which a single quotient digits is calculated) of 4ns. The maximally redundant SRT [10] follows with a critical path of 6.2ns and the slowest implementation is the minimally redundant SRT architecture [4] with 7.1ns.

In the Tables 2 - 4, the results of the power estimation are shown for a frequency of 200 MHz. The tables are divided into the power estimation for the divider architecture and the additional hardware for the square-root implementation. All elements are simulated separately. However, the load capacity of the next stage and wiring capacity between the two blocks are considered in the power simulations. Registers have been implemented after each iteration for pipelining. The total power consumption of the three presented architectures are P_{GST}=141.4mW, $P_{SRT,mr}$=102.6mW and $P_{SRT,MR}$=116.8mW, respectively, all at the frequency of 200MHz. The overall latency of the GST is 35% smaller compared to the maximally redundant SRT implementation, and 44% smaller compared to the minimally redundant SRT implementation. Alternatively, by fixing the iteration latency to 6.2 ns and 7.1ns for the minimally-redundant

Table 2. Power estimation of the minimally redundant GST architecture

Minimally redundant GST architecture	
K-selector	0.3mW
Full-adder	0.07mW · 14 · 2 = 2mW
Y-MXA	2.4mW
X-MXA	2.4mW
Register	0.04mW · 28 · 13 = 14.5mW
Register	0.04mW · 42 · 13 = 21.8mW
random logic for RW	0.1mW · 12 = 1.2mW
random logic for DC	0.2mW · 12 = 2.4mW
Muxes,Hybrid-Adder	0.325mW · 14 · 12 = 54.6mW
On-the-fly converter	0.041mW · 13 = 0.5mW
sub-total	102.1mW
random logic for DC'	0.2mW · 12 = 2.4mW
Muxes	0.021mW · 24 = 0.5mW
Muxes	0.021mW · 26 · 2 = 1.2mW
Muxes	0.021mW · 28 · 3 = 1.8mW
Muxes	0.014mW · 8 · 13 = 1.5mW
Muxes	0.014mW · 8 · 13 = 1.5mW
Muxes	0.014mW · 6 · 13 = 1.0mW
Register	0.04mW · 3 · 13 = 1.5mW
Muxes	0.014mW · 6 · 13 = 1.0mW
5-bit CRA	0.425mW · 4 · 13 = 22.1mW
7-bit MXA	0.592mW ·13 = 7.7mW
total	141.4mW

Table 3. Power estimation of the minimally redundant SRT architecture

Minimally redundant SRT architecture	
Converter	0.361mW · 13 = 4.7mW
XOR	0.025mW · 8 · 13 = 2.6mW
OR	0.015mW · 6 ·13 = 1.2mW
PLA	0.1mW · 13 = 1.3mW
Register	0.04mW · 28 · 13 = 14.5mW
Register	0.04mW · 42 · 13 = 21.8mW
Muxes,Hybrid-Adder	0.325mW · 14 · 12 = 54.6mW
On-the-fly converter	0.041mW · 13 = 0.5mW
sub-total	101.2mW
Muxes	0.014mW · 4 · 13 = 0.7mW
Muxes	0.014mW · 4 · 13 = 0.7mW
total	102.6mW

Table 4. Power estimation of the minimally redundant SRT architecture

Maximally redundant SRT architecture	
MXA 7bit	0.507mW·13 = 6.6mW
XOR	0.025mW · 7 · 13 = 2.3mW
OR	0.015mW · 6 ·13 = 1.2mW
NS	0.2mW · 13 = 2.6mW
LS	0.1mW · 13 = 1.3mW
DD	0.07mW · 13 = 0.9mW
Register	0.04mW · 24 · 13 = 12.5mW
Register	0.04mW · 48 · 13 = 25.0mW
Muxes	0.03mW · 24 · 13 = 9.4mW
Adder	0.07mW · 24 · 13 = 21.8mW
Muxes	0.03mW · 24 · 13 = 9.4mW
Adder	0.07mW · 24 · 13 = 21.8mW
On-the-fly converter	0.041mW · 13 = 0.5mW
sub-total	115.4mW
Muxes	0.014mW · 4 · 13 = 0.7mW
Muxes	0.014mW · 4 · 13 = 0.7mW
total	116.8mW

and maximally-redundant SRT architecture, respectively, the supply voltage of the GST architecture can be reduced to obtain a low power implementation. Assuming $V_{dd} = 3.3V$, $V_t = 0.67V$, $\beta^2_{GST} = 0.5802$ and $\beta^2_{GST} = 0.4977$, respectively, are obtained. The supply voltage may accordingly be reduced to $V_{dd,GST}$=2.51V by comparing the GST to the maximally redundant SRT and to $V_{dd,GST}$=2.32V by comparing the GST to the minimally redundant SRT architecture. Hence the total power consumption of the GST-SQRT architecture with same latency as the maximally redundant SRT-SQRT, is P_{GST}=66mW, and P_{GST}=47.6mW compared to the minimally redundant SRT (see Tables 5 and 6). This corresponds to 20% and 40% less power, respectively. Another way of showing the superiority of the GST over the SRT is by calculating the Power-Delay and Energy-Delay products of the circuits. Those results are also shown in Tables 5 and 6, respectively. Many improvements to the divider implementations have been suggested in [5]. Since these improvements are applicable to both SRT and GST dividers in an identical way, so these don't change the overall ratio between the GST and SRT behavior of speed and power.

7 Conclusion

For the first time, the GST division algorithm has been successfully applied to the square-root operation in a hardware-efficient manner. The additional hardware costs increase the power consumption and the critical path only slightly, so that the benefits of the GST algorithm over the SRT algorithm are maintained. Simulations have shown that the overall latency of the GST is 35% smaller compared to the SRT. Alternatively, by fixing the latency, the GST division requires 42% less and the GST square-root requires 20% less power compared to the SRT using a maximally redundant radix-4 digit set. To conclude, the GST approach leads to a superior design for division, square-root and shared division/square-root architectures for la-

Table 5. Comparison between GST and SRT_{MR} Algorithm

	GST	SRT_{MR}
Latency	52ns	79.3ns
Power$_{SQRT@4ns}$	176.7mW	-
Power$_{Division@4ns}$	127.6mW	-
Power$_{SQRT@6.2ns}$	114mW	82.7mW
Power$_{Division@6.2ns}$	82mW	81.6mW
Power$_{SQRT}\cdot$ Delay	7.1nJ	5.1nJ
Power$_{Division}\cdot$ Delay	5.1nJ	5nJ
Energy$_{SQRT}\cdot$ Delay	284nJ ns	316nJ ns
Energy$_{Division}\cdot$ Delay	204nJ ns	310nJ ns
$\beta^2\cdot$ Power$_{SQRT@6.2ns}$	66mW	82.7mW
$\beta^2\cdot$ Power$_{Division@6.2ns}$	47.6mW	81.6mW
Latency	0.65	1
Power$_{SQRT@4.4ns}$	1	-
Power$_{Division@4.4ns}$	1	-
Power$_{SQRT@6.2ns}$	1	0.72
Power$_{Division@6.2ns}$	1	1
Power$_{SQRT}\cdot$ Delay	1	0.72
Power$_{Division}\cdot$ Delay	1	1
Energy$_{SQRT}\cdot$ Delay	0.90	1
Energy$_{Division}\cdot$ Delay	0.66	1
$\beta^2\cdot$ Power$_{SQRT@6.2ns}$	0.80	1
$\beta^2\cdot$ Power$_{Division@6.2ns}$	0.58	1

Table 6. Comparison between GST and SRT_{mr} Algorithm

	GST	SRT_{mr}
Latency	52ns	92.3ns
Power$_{SQRT@4ns}$	176.7mW	-
Power$_{Division@4ns}$	127.6mW	-
Power$_{SQRT@7.1ns}$	99.5mW	82.3mW
Power$_{Division@7.1ns}$	71.9mW	81.3mW
Power$_{SQRT}\cdot$ Delay	7.1nJ	5.8nJ
Power$_{Division}\cdot$ Delay	5.1nJ	5.8nJ
Energy$_{SQRT}\cdot$ Delay	284nJ ns	415nJ ns
Energy$_{Division}\cdot$ Delay	204nJ ns	410nJ ns
$\beta^2\cdot$ Power$_{SQRT@7.1ns}$	49.5mW	82.3mW
$\beta^2\cdot$ Power$_{Division@7.1ns}$	35.8mW	81.3mW
Latency	0.56	1
Power$_{SQRT@4.4ns}$	1	-
Power$_{Division@4.4ns}$	1	-
Power$_{SQRT@7.7ns}$	1	0.83
Power$_{Division@7.7ns}$	0.88	1
Power$_{SQRT)}\cdot$ Delay	1	0.83
Power$_{Division}\cdot$ Delay	0.88	1
Energy$_{SQRT}\cdot$ Delay	0.68	1
Energy$_{Division}\cdot$ Delay	0.51	1
$\beta^2\cdot$ Power$_{SQRT@7.7ns}$	0.60	1
$\beta^2\cdot$ Power$_{Division@7.7ns}$	0.44	1

tency and power critical applications.

References

[1] I. Abu-Kather, A. Bellaouar, and M. Elmasry. Circuit Techniques for CMOS Low-Power High Performance Multipliers. *IEEE Journal of Solid-State Circuits*, 31(10):1535–1546, Oct. 1996.

[2] A. Avizienis. Signed-Digit number representations for fast parallel arithmetic. *IRE Trans. Electron. Comp.*, EC-10:389–400, Sept. 1961.

[3] A. Chandrakasan, S. Sheng, and R. Brodersen. Low power CMOS digital design. *IEEE Journal of Solid State Circuits*, 27:473–485, April 1992.

[4] J. Fandrianto. Algorithm for high speed shared radix-4 division and radix-4 square root. In *IEEE 1987 8^{th} Symposium on computer arithmetic*, pages 73–79, Como, Italy, May 1987.

[5] D. Harris, S. Oberman, and M. Horowitz. SRT Division architectures and implementations. In *Proc. 13^{th} Symp. Computer Arithmetic*, pages 18–24, July 1997.

[6] I. Koren. *Computer Arithmetic Algorithms*, chapter 7-8. Prentice Hall, New Jersey, USA, 1993.

[7] T. Lang and P. Montuschi. Higher Radix Square-Root with Prescaling. *IEEE Trans. on Computer*, 41(12):1606–1611, Dec. 1992.

[8] L. Montalvo. *Number systems for high performance dividers, PhD Thesis*. Univ. of Grenoble, France, 1995.

[9] L. A. Montalvo, K. K. Parhi, and A. Guyot. New Svoboda-Tung Division. *IEEE Trans. on Computer, to appear*, 1998.

[10] P. Montuschi and L. Ciminiera. Design of a radix 4 division unit with simple selection table. *IEEE Trans. on Computers*, 41(12):1606–1611, Dec 1992.

[11] K. K. Parhi. Fast low-Energy VLSI binary addition. In *Proc. of IEEE Int. Conf. on Computer Design, (ICCD)*, pages 676–684, Austin, TX, Oct. 1997.

[12] D. S. Phatak and I. Koren. Hybrid signed-digit number systems: a unified framework for redundant number representations with bounded carry propagation chains. *IEEE Trans. on Comp.*, 43(8):880–891, Aug. 1994.

[13] J. Robertson. A new class of digital division methods. *IRE Trans. Electron. Comp.*, EC-7:218–222, Sep. 1958.

[14] A. Svoboda. An algorithm for division. *Information Processing Machines (Prague, Czech Republic)*, (9):25–32, 1963.

[15] N. Takagi, H. Yasuura, and S. Yajima. High-speed VLSI multiplication algorithm with a redundant binary tree addition. *IEEE Trans. on Comp.*, C-34(9):789–796, Sept 1985.

[16] K. Tocher. Techniques of multiplication and division for automatic binary computers. *Quart. Journ. Mech. Appl. Math.*, 2(3):364–384, 1958.

[17] C. Tung. A division algorithm for signed-digit arithmetic. *IEEE Trans. Computers (short notes)*, C-17:887–889, Sept. 1968.