Design of a fast radix-4 SRT divider and its VLSI implementation

C.-L. Wey and C.-P. Wang

Abstract: The design of a fast divider is an important issue in high-speed computing. The paper presents a fast radix-4 SRT division architecture. Instead of finding the correct quotient digit, an estimated quotient digit is first speculated. The speculated quotient digit is used to simultaneously compute the two possible partial remainders for the next step while the quotient digit is being corrected. Thus, this two-step process does not influence the overall speed. Since the decision-making circuits can be implemented with simple gate structures, the proposed divider offers fast speed operation. Based on the physical layout, the circuit takes 247 ns for a double precision division (56 bits for fraction part), where the 2 μm CMOS technology in MAGIC is employed and simulated.

1 Introduction

The design of fast dividers is an important issue in high-speed computing because division accounts for a significant fraction of the total arithmetic operation [1]. Most implementations for the division are based on the SRT algorithm that uses a recurrence producing one quotient digit for each step [2–9]. The speed of such SRT-based algorithms that use a recurrence producing one quotient digit for each step [2–9]. The speed of such SRT-based dividers is mainly determined by the complexity of the quotient-digit selection. Fig. 1 illustrates an architecture of a radix-4 SRT divider which employs a quotient-digit selection table (QST). The use of QST significantly reduces the complexity of quotient-digit selection. However, the table size increases drastically with high radices [2, 9]. The table size can be reduced significantly by estimating the quotient digit instead of finding the exact one [9]. The estimated quotient digit is calibrated in parallel with updating the new partial remainder. Since the two-step process does not affect the division speed, the approach has fast speed performance due to the significant reduction in table size. This paper presents the detailed design of a fast radix-4 SRT division and its VLSI implementation. Results show that, based on the physical layout, the circuit takes 291 ns to compute a double precision division (56 bits in the fraction part), where the CMOS technology in MAGIC is employed for simulation.

2 Radix-4 division

Consider the following recursive equation for the partial remainder in a high-radix SRT division [2],

\[ r_i = r_{i-1} - q_i D, \]

where \( b = 2^m \) is the radix, \( D \) is the divisor, \( r_{i-1} \) is the partial remainder at the \( (i-1) \)th step, and \( q_i \) is the \( i \)th quotient digit. The high-radix SRT division can be implemented in such a way that its quotient digit \( q_i \) is selected from a digit set \( \{-x, \ldots, -1, 0, 1, \ldots, x\} \), where \( \left( \frac{b-1}{2} \right) \leq x \leq \frac{b-1}{2} \). The ratio \( \kappa = \frac{x}{(b-1)} \) is a measure of the redundancy in the representation of the quotient digits. The smaller the value of \( \kappa \), the smaller is the redundancy in the number system for the quotient.

Fig. 1b shows the \( P - D \) plot of a radix-4 division [2], where \( (\beta, \alpha) = (4, 2) \), \( \kappa = 2/3 \) and \( D = [D_{\text{min}}, D_{\text{max}}] = [0.5, 1.0] \). The quotient digit \( q \in \{-2, -1, 0, 1, 2\} \). Let \( P = 4r_{i-1} \) denote the previous partial remainder, where \( r_{i-1} \) is the partial remainder at the \( (i-1) \)th step and \( |r_{i-1}| \leq (2/3)D \). Thus, the upper limits and lower limits for \( P \) are

\[ (-2/3 + q)D \leq P \leq (2/3 + q)D \tag{1} \]

Fig. 1 Radix-4 SRT division

\( a \) Architecture; \( b \) P-D plot and stair function for \((\beta, \alpha) = (4, 2)\), where shaded areas are overlapping regions; \( c \) Regions for various partial remainders \( P \); \( d \) Overlapping regions

© IEE, 1999
IEE Proceedings online no. 19990524
DOI: 10.1049/ip-cdt:19990524
Paper first received 13th November 1997, and in revised form 21st April 1999
The authors are with the Department of Electrical Engineering, Michigan State University, East Lansing, MI 48824-1226, USA


205
and the regions for all $qs$ are listed in Fig. 1c, where $0.5 \leq D < 1$. There exists an overlapping region between two adjacent regions corresponding to two consecutive values of a quotient digit, as the shaded areas show in Fig. 1b. Let $P_{j+1}$ denote the overlapping region for $q = j$ and $q = j + 1$. The overlapping region $P_{j+1}$ is expressed as

$$P_{j+1} = ((P, D)(-2/3 + j + 1)D \leq P \leq (2/3 + j)D)$$

Thus, the regions for all $qs$ are listed in Fig. 2a, and the regions for all $qs$ is shown in Fig. 2d. The quotient digit selection table stores the quotient assignment in the partial remainder to select the quotient digit. This shows that the quotient digit is estimated by

$$P = \left\{ \begin{array}{ll}
1 & \text{if } c_0 \leq P \leq c_1 \\
-1 & \text{if } c_{-1} \leq P \leq c_0 \\
0 & \text{if } P \leq c_{-1}
\end{array} \right.$$  

Once the estimated quotient digit $q^\#$ is determined, the range of the partial remainder is

$$(2/3)D \leq P^* = P - q^\#D \leq (5/3)D$$

and the correction value $q^*$ is determined as follows:

$q^* = 1$ if $(1/3)D \leq P^* \leq (5/3)D$ and $q^* = 0$ if $(-2/3)D \leq P^* \leq (2/3)D$. This shows that $P^*$ is equivalent to $P$ in Eqn. 1 with $q = 0$ and $q = 1$. Fig. 2b illustrates the $P^* - D$ plot, which is exactly the same as the $P - D$ plot in Fig. 1b for $q = 0$ and 1. Therefore, two comparison constants are used to determine $q^*$. They are $c_0 = (0.01)$ for $D = [0.10, 0.11]$ and $c_1 = (0.10)$ for $D = [0.11, 1.00]$, as shown in Fig. 2d, and

$$q^* = \left\{ \begin{array}{ll}
1 & \text{if } c_0 \leq P^* \leq D = [0.10, 0.11] \\
0 & \text{if } c_1 \leq P^* \leq D = [0.11, 1.00] \\
0 & \text{otherwise}
\end{array} \right.$$  

or

$q^* = 1$ if $(c_2 \leq P^*D < 0.11)$ or $c_1 \leq P^*$

(7)

Once $q^*$ is determined, the actual quotient digit $q = q^\# + q^*$ and the partial remainder is $P^k$ if $q^* = 0$, or $P^d$ if $q^* = 1$. The detailed division process is summarised in Algorithm 1, and Fig. 3 illustrates the stepwise procedure with an example, where the dividend $X = (0.01111111)n$, the divisor $D = (0.1001)n$, and $(\beta, \alpha) = (4, 2)$. The division process starts with comparing $X$ to the comparison constants in Fig. 2a to generate $q^* = 1$ so that $r_0 = X - D$. Since $X < D$, this results in $q^* = 0$. Thus, $q_0 = q^* + q^* = 1$. It is followed by the generation of $q_1$, where a speculated quotient digit

$q_1 = 1$ is first estimated. Together with $q_1 = 1$, we obtain the actual quotient $q_1 = 0$, which is the same as $q_1 = 0$. 

Fig. 2 Proposed radix-4 SRT division

a Estimated quotient digit selection; b Correction value selection; c Regions of partial remainderers for all estimated quotient digits; d Overlapping regions for estimated quotient digits

$$c_1 = (000.1), c_0 = 0 \text{ and } c_{-1} = (111.1).$$

Thus, the quotient digit is estimated by

$$q^\# = \left\{ \begin{array}{ll}
1 & \text{if } c_1 \leq P \\
0 & \text{if } c_0 \leq P \leq c_1 \\
-1 & \text{if } c_{-1} \leq P \leq c_0 \\
-2 & \text{if } P \leq c_{-1}
\end{array} \right.$$  

(5)

3 Proposed radix-4 division

Instead of selecting the correct quotient digit $q$, $x \leq q \leq z$, the proposed approach first estimates a quotient digit $q^\#$, $x \leq q^\# \leq z - 1$, such that $q \in \{q^\#, q^\# + 1\}$, i.e. the actual quotient digit is either $q^\#$ or $q^\# + 1$, and the possible partial remainder is either $P^0 = 4r_{-1} - q^\#D$ or $P^1 = 4r_{-1} - (q^\# + 1)D$. Thus, the actual quotient digit $q = q^\# + q^*$, where $q^*$, referred to as a correction value, is either 0 or 1. This division process includes two steps: (i) quotient digit estimation, and (ii) possible remainder updating and quotient digit estimation. This Section first describes the proposed division algorithm, hardware implementation and error analysis. In addition, the speed performance of the proposed architectures is estimated. To actually estimate the performance, the physical layout has been simulated.

3.1 Algorithm development

The estimated quotient digit $q^\#$ is selected as follows. For an estimated quotient digit $q^\#$, the upper and lower limits for the corresponding partial remainder $P$ are

$$(-2/3 + q^\#)D \leq P \leq (2/3 + q^\# + 1)D$$

(4)

Thus, the regions for all $q^\#$s are listed in Fig. 2a, and the overlapping regions are shown in Fig. 2b. Fig. 2a shows that the new overlapping regions are much wider and flatter than those in Fig. 1b, where $P$ is represented in a two’s-complement form. Three comparison constants can be generated from the overlapping regions. They are

$$P_{j+1} = ((P, D)(-2/3 + j + 1)D \leq P \leq (2/3 + j)D)$$

(2)

Therefore, the overlapping regions in Fig. 1b are listed in Fig. 1d. The quotient digit in the overlapping region $P_{j+1}$ can be either $q = j$ or $q = j + 1$. The implication of having an overlap region is that we have a choice of values, of both the partial remainder and the divisor, that will eventually separate these two adjacent regions. The selected value of the partial remainder and divisor separating the adjacent regions of $q$ will serve as comparison constants during the execution of the divide operation. If there is a value $c$ satisfying

$$(j + 1/3)D_{\min} \leq c \leq (j + 2/3)D_{\min}$$

then the selection of $q$ will be independent of $D$ and will depend only on $P$. The number of bits required to represent this constant determines the necessary precision when examining the partial remainder to select the quotient digit $q$. However, if this inequality is not satisfied, the interval $[D_{\min}, D_{\max}]$ is partitioned into several smaller intervals. The stepping points determine the precision (i.e. the number of bits) at which we examined $D$, while the height of the steps determines the precision at which the partial remainder has to be examined. Therefore, the quotient-digit selection table stores the quotient assignment in the $P - D$ plot and the table can be implemented by either a 1.5Kbit ROM or ($n_a, n_b, n_c$) = (9, 3, 35) – PLA [9], where $n_a$, $n_b$, $n_c$ are the number of inputs, outputs and product terms, respectively.

Algorithm 1.
Step 1. (Speculated quotient digit estimation)
1.1 IF (start = 0) THEN (P = X; i = 0; start = 1) ELSE P = 4 × r1; 1.2 Generate q* from eqn. 5;
Step 2. (Quotient digit correction)
2.1 Calculate P0 = P − q*D and P1 = P(q* + 1)D;
2.2 Generated q* from eqn. 7;
2.3 q = q* + q†.
2.4 IF (q* = 0) THEN r1 = P0 ELSE r1 = P1;
2.5 i = i + 1; GOTO Step 1.

a

\[ X = 0.01111111 \text{ and } D = 0.1001 \]
\[ X = 0.01111111 \]
\[ \text{Add -D } r_1 = 111.111111 \]
\[ \text{Add-2D } r_1 = \frac{111.111111}{101.011111} \]
\[ 4r_1 \]
\[ 111.111111 \]
\[ S = \frac{1}{1} \text{ and } r_0 = r_0' \]
\[ \text{Add } D, r_1 = \frac{000.000001}{111.111111} \]
\[ \text{Add } 0, r_1 = \frac{111.111111}{111.111111} \]
\[ 4r_1 \]
\[ 110.111111 \]
\[ S = \frac{1}{1} \text{ and } r_0 = r_0' \]
\[ \text{Add } -D, r_1 = \frac{000.000000}{111.111000} \]
\[ \text{Final quotient } Q = (0.0123) = (0.100010) = (0.111) \_2 \]

b

Fig. 3 Radix-4 division
a Proposed algorithm; b Example

For q2, a speculated digit q2 = −2 is estimated. With q2 = 0, we obtain q2 = −2. Therefore, the final quotient Q = (0.1110) and the remainder R = (0.0001) × 2⁻⁴ = 1. It can be easily verified that X = Q*D + R.

3.2 Hardware implementation
Based on Algorithm 1, Fig. 4 illustrates the proposed architecture. It is assumed that the divisor N is an n-bit positive normalised binary number, where D = (0.1 d₂ d₃ ... dₙ)₂, the dividend X is a k-bit binary number ranged between −(8/3)D and (8/3)D, where X is represented as (SPᵢ P₀ P₋₁ P₋₂ ... P₋ₙ)₂ in two’s complement form and S is the sign bit, i.e., S = 0, 1, 2, n, and the quotient Q = (q₀ q₁ q₂ ... qₙ)₂, where qᵢ ∈ {−2, −1, 0, 1, 2} is a binary redundant digit, i = 0, 1, 2, ... n. At the first cycle, the first n + 3 bits of X, i.e., (SPᵢ P₀ P₋₁ P₋₂ ... P₋ₙ)₂ are processed, and in the following cycles, two new bits of X are shifted into the updated remainder.

According to Algorithm 1, we first estimate the quotient digit q* from Block MH. The multiplexer circuit takes the quotient digit q* and generates −q*D and −(q* + 1)D. Two adders are used to compute P₀ = 4r₋₁ − q*D and P₁ = 4r₋₁ + [(q* + 1) D] in Step 2.1 and the results are compared with the constants shown in Fig. 2 to generate q* and q†, where only the seven most significant bits of P₀ and P₁ (to be explained shortly) are required. Two parallel addition schemes without carry-propagation delay may be considered: signed-digit adder [2] and carry-save adder (CSA). In practice, CSA is preferred when the adder is used to perform both addition and subtraction. Therefore, this implementation uses two (n + 3)-bit CSAs and two 7-bit CLAs (carry-lookahead adders) to avoid long carry propagation. After generating the correction value q* in Step 2.2, two multiplexers are used to respectively select the CSA and CLA outputs for P₀ or P₁. The process will be repeated for n times. In this implementation a signal start is used to indicate the beginning of the division process. More specifically, when the division begins, i.e., start = 0, Block QH selects the estimated quotient digit q* for the dividend X, and the signal start is set to a 1, and then the CSAs compute P₀ = X − q*D and P₁ = X − (q* + 1)D. The remaining procedure is the same as that described above. Therefore, for an n-bit division, the process needs n + 1 cycles.

3.2.1 Block QS: Block QS generates the correction value q* from the computed remainder P₀. Given an updated partial remainder 4r₋₁ and the estimated quotient digit q*, by eqn. 7, q* is determined by the comparison constants in Fig. 2d. Let P₀ = (SPᵢ P₀ P₋₁ ... P₋ₙ)₂. Since the maximum positive value of P₀ is 5/3 and the bit P₁ = 0 is true for all positive P₀, the bit P₁ is ignored here. The comparison constants shown in Fig. 2d can be tabulated as shown in Table 1. For a P₀, c₃ ∈ [000.10) ≤ P₀, it can be represented by (SPᵢ P₀ P₋₁ ... P₋ₙ)₀ = (000.1 x x x) or (000.1 x x x), and will be truncated as (SP₀ P₋₁ P₋₂)₀ = (001.x) or (000.1). Note that the bit P₁ is always 0 for a positive P₀. Thus, the statement ‘c₃ ≤ P₀’ in eqn. 7 is equivalent to ‘S = 0 & [P₀ = 0 or P₋₁ = 1]’. Similarly, the statement ‘c₂ ≤ P’ is equivalent to ‘S = 0 & [P₀ = 0 or P₋₁ = 1 or P₋₂ = 1]’. Since D ∈ [0.5, 1], a value D < 0.11 is expressed as D = (0.1 d₂ d₃ ... dₙ)₂ = (0.10xx ... x). Thus, ‘D < 0.11’ is equivalent to ‘d₋₂ = 0’. Thus, eqn. 7 can be rewritten as ‘q* = 1 if [S = 0 & [P₀ = 0 or P₋₁ = 1]]’ or [S = 0 & [P₀ = 1 or P₋₁ = 1 or P₋₂ = 1] & d₋₂ = 0].’

Table 1: Comparison Constants for q*  

<table>
<thead>
<tr>
<th>q*</th>
<th>d₋₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>01.xx</td>
<td>1</td>
</tr>
<tr>
<td>00.10</td>
<td>1</td>
</tr>
<tr>
<td>00.01</td>
<td>1</td>
</tr>
<tr>
<td>00.00</td>
<td>0</td>
</tr>
<tr>
<td>1x.xx</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig. 5a shows the logic implementation of eqn. 8.

3.2.2 Block QH: The correction value q* selects the updated remainder rᵢ from either P₀ or P₁, and the first seven most significant bits of the updated remainder. As shown in Fig. 4, the updated remainder is shifted left by 2 bits to block QH for generating q*. Thus, after shifting 2
represents the binary representation of the absolute value of $q_i$ the following logic function:

$$q^* = q', q^1 = q'(q^* \oplus q^*)$$ and $q^0 = q' \oplus (q' \oplus q^*)$ \hspace{1cm} (10)$$

Fig. 5c shows the logic implementation of eqn. 10.

### 3.2.4 Adders–CLA and CLA:

Two CSAs (carry-slave adders) are used to compute the remainders $P_i^0$ and $P_i^1$. The CSA with shifting is realised by FAs and latches, as shown in Fig. 5d. The sum and carry outputs of the first seven FAs are fed to a 7-bit CLA.

### 3.2.5 Multiplexer circuitry:

The multiplexer circuitry (MUXC), as shown in Fig. 6 generates $-q^*D$ and $-(q^* + 1)D$, which are chosen from $\{-2D, 0, 1, 2D\}$. Since the divisor $D$ is an $n$-bit positive normalised binary number, the numbers $-D$ and $-2D$ are represented in two’s complement form. More specifically, let $D$ be denoted as $(S_Dd_0d_{-1}d_{-2}\ldots d_{-n})$, where $S_D=1$ is the sign bit, $d_0=0$ and $d_{-1}=1$. Then, $-D = ((S_D-d_0)e_{-1}e_{-2}\ldots e_{-n})$ and $-2D = ((S_D)e_{-1}e_{-2}\ldots e_{-n}0)$, where $e_i = d_i'$ and $e_{-n} = d_{-n} + 1$. Note that the $(-n)$th bit of $(2D)$, or, namely, $e_{(-n+1)}$ is a 0. The two’s complement form is realised by using a simple one’s complementation and assigning a 1 as the initial carry of the CSA.

The estimated quotient digit $q^\circ \in \{1, 0, -1, -2\}$, is represented by two bits, i.e. $q^\circ = (q^1, q^0)$, and its values, 1, 0, $-1$ and $-2$, are encoded as $(0, 1), (0, 0), (1, 0)$ and $(1, 1)$, respectively, as shown in Table 4. Fig. 6a illustrates the block diagram that implements the MUXC. Each slice of the MUXC is realised by six 2-to-1 multiplexers (MUXs), as shown in Fig. 6b. The MUXC is fed to the CSA, as shown in Fig. 7a. To generate the initial carry of CSA, the inputs to the $-n$th bit of

### Table 2: Comparison Constants for $q^\circ$

<table>
<thead>
<tr>
<th>$SP_{-1}P_{-2}P_{-3}$</th>
<th>011.1</th>
<th>01x.x</th>
<th>001.x</th>
<th>000.1</th>
<th>000.0</th>
<th>111.1</th>
<th>111.0</th>
<th>110.x</th>
<th>10x.x</th>
<th>100.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q^1$</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>$q^0$</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
</tr>
</tbody>
</table>

The following logic functions realise the truth table

$S_q = q'$, $q^1 = q'(q^* \oplus q^*)$ and $q^0 = q' \oplus (q' \oplus q^*)$ \hspace{1cm} (10)

Fig. 5c shows the logic implementation of eqn. 10.

### Table 3: Truth table for block QC

<table>
<thead>
<tr>
<th>$q^\circ=0$</th>
<th>$q^\circ=1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q^1$</td>
<td>$q^0$</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table 4: Function MUXC and bit assignment for $(-n)$ bit of CSA

<table>
<thead>
<tr>
<th>$q^1$</th>
<th>$q^0$</th>
<th>$-q^*D$</th>
<th>$-(q^* + 1)D$</th>
<th>$y_{-n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-2D</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-D</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>$D$</td>
<td>0</td>
</tr>
<tr>
<td>-2</td>
<td>1</td>
<td>1</td>
<td>2D</td>
<td>0</td>
</tr>
</tbody>
</table>

MUXC are modified by assigning a 0 to \( d_i + 1 = d_{i+1} \) and a 1 to \( e_j + 1 = e_{j+1} \). The inputs to the last bit of CSA are modified as follows: the initial carry of CSA is fed to \( y_{n-1} \) of the \( n \)th FA, where \( y_{n-1} = 1 \) (1) if the CSA is used as an adder (subtractor). Therefore, for \(-q \) to \( D \), \( y_{n-1} = 0 \) for \( 0 \) and \( D \), and \( y_{n-1} = 1 \) for \(-D \), as shown in Table 4. On the other hand, for \(-D \) to \( 2D \), it is necessary to add the initial carry ‘1’ to the carry-in bit of the \((n-1)\)th bit slice. Without modifying the \((n-1)\)th bit slice, this can be achieved by setting \( y_{n-1} = 1 \) and \(-D \) to \( -D \), as shown in Fig. 6a. Therefore, the logic function for \( y_{n-1} \) is \((q^s) \) and \((q^t) \) for \(-q \) to \( D \) and \(-q \) to \( 2D \), respectively.

3.2.6 VLSI implementation—speed and area: Fig. 8 shows the physical layout of an appropriate 56-bit floating-point divider (fraction part). The layout is generated by the MAGIC layout editor, where 2 \( \mu \)m SCMOS technology is employed. The layout includes a 57-bit CSA, two 7-bit CLAs with 1-level, a 57-bit MUXC with 4-to-1 MUXs, and BLOCK QS, QC and QH. There, different types of MUX circuits are used. Type-1, MUX1, includes the one receiving its inputs from CLAs and the one sending its outputs to QH; type-2, MUX2, includes the one receiving its inputs from CSAs and the one sending the outputs to CSAs; and type-3, MUX3, is the MUXC. The propagation delay time of each unit has been simulated by Pspice, where the circuit parameters are extracted from the layout. Table 5 lists the size and propagation delay of each unit. According to the layout in Fig. 8 with the routing areas, for 56-bit radix-4 SRT division, the total area is approximately \( 3.7 \times 5.3 \) mm\(^2\), and its delay time is 13.9 ns per cycle. The division time can be improved by the alternative architecture shown in Fig. 9 where an extra MUXC is used. As indicated by the bold lines, Block QS is no longer in the critical path. Thus, the critical path includes the MUX3, MUX2, CSA, CLA, MUX1 and QH. Simulation results show that the circuit has a delay of 10.4 ns per cycle, or 291.2 ns for the 56-bit division. However, the speed improvement is achieved at the increased cost of area which is nearly \( 4 \times 5.3 \) mm\(^2\), as shown in Fig. 10. Since block QH is on the critical path, improving its speed performance will also make the division process faster. Eqn. 9 shows that the output \( q^s \) is a function of \( P_{-1}, P_{-2} \) and \( P_{-3} \). Therefore, the output \( q^s \), as a function of three variables, can be realised by a 4-to-1 MUX which takes only 0.43 ns. As a result, the total delay in the critical path is 8.83 ns. Therefore, the proposed SRT algorithm can be achieved with a delay of 247.24 ns for 56-bit division, where 2 \( \mu \)m CMOS technology is employed.
Table 6: Double precision division performance comparison

<table>
<thead>
<tr>
<th>Chip</th>
<th>Fraction only time, ns</th>
<th>Radix</th>
</tr>
</thead>
<tbody>
<tr>
<td>W3364 (weitek)</td>
<td>675</td>
<td>4</td>
</tr>
<tr>
<td>R3010B (MIPS)</td>
<td>560</td>
<td>4</td>
</tr>
<tr>
<td>Method-B [10]</td>
<td>238</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 6 shows the performance comparison given in [10], where 1.2 μm CMOS technology is employed for all cases. Comparing the proposed radix-4 division, 247.24 ns with 2 μm CMOS technology, to the performance of various designs shown above, the proposed circuit performance is promising. The proposed circuit can be further improved by using 1.2 μm or better technology.

4 Conclusion

This paper presents a simple yet fast radix-4 divider design and its VLSI implementation. The proposed division method first calculates a quotient digit. The speculated digits are used to compute the two possible partial remainders, for the next step, in parallel with the quotient-digit correction process. The algorithm can be implemented with only a delay of 247.24 ns for double precision division. Although this paper is presented only for radix-4 division, the same design concept is readily extended for high radices to reduce the quotient-digit selection table size and make the use of high-radices become possible and practical [9]. Also, similarly to the discussion in [10], the proposed architecture can also be implemented for square-root calculation. It should be mentioned that the primary focus of the developed algorithm and hardware implementation was placed on optimising execution. It is worthwhile to further investigate and develop an alternative design and implementation which is optimised in terms of execution time, chip area and/or power dissipation.

5 References


6 Appendix: Error analysis

This Appendix is to justify that a 7-bit CLA is sufficient for this implementation. Let $c_i$ and $s_i$, $j = 1, 0, -1, \ldots, -n$, denote the $j$th carry and sum bits of the CSA for the remainder $P^{b_1}$ at the $j$th cycle, respectively. Generating the complete $(n+3)$ bit binary value for $P^{b_1} = (SP_1, P_0, P_{-1}, P_{-2}, \ldots, P_{-n})$ requires an $(n+3)$-bit CLA to sum the carry and sum bits as follows:

$$Ss S_1 S_0 S_{-1} S_{-2} S_{-3} S_{-4} \ldots S_{-(n-1)} S_{-n}$$

$$+ C_1 C_0 C_{-1} C_{-2} C_{-3} C_{-4} C_{-5} C_{-6} \ldots C_{-n} 0$$

$$S P_1 P_0 P_{-1} P_{-2} P_{-3} P_{-4} \ldots P_{-(n-1)} P_{-n}$$

To avoid long carry propagation, this implementation rounds the value of $P^{b_1}$ to the bit $P_{-n}$, i.e.

$$Ss S_1 S_0 S_{-1} S_{-2} S_{-3} \ldots S_{-t}$$

$$+ C_1 C_0 C_{-1} C_{-2} C_{-3} C_{-4} \ldots C_{-(t+1)}^*$$

$$S P_1 P_0 P_{-1} P_{-2} P_{-3} \ldots P_{-t}$$

where $C_{-(t+1)} = C_{-(t+1)} + C^n$. The rounding scheme defines the value of $C^n$, where $C^n = 0$ if $C_{-(t+2)} = S_{-(t+1)} = 0$, and $C^n = 1$, otherwise. The signal $C_{in}$ ORs both $S_{-(t+1)}$ and $C_{-(t+2)}$. Note that $P_3$ is not connected to block QH nor block QS.

Let $P_0^{b_1}$ and $P_{ro}^{b_1}$ denote the rounded and truncated values of $P^{b_1}$. If $C^n = 0$, then $P_0^{b_1} = P_{ro}^{b_1}$ and thus no error results. On the other hand, if $C^n = 1$, i.e. $P_0^{b_1} \neq P_{ro}^{b_1}$, then two cases can be identified: both values result in either the same $q^i$ and $q^i_*$, or different $q^i$ and $q^i_*$, where $q^i_*$ and $q^i$ are the $r$th estimated quotient digit and the corrected value, respectively. More specifically, in the former case, both values may be different, but they may result in the same $q^i$ and $q^i_*$. Thus, both values generate the same quotient digit $q$ and no error results. For the latter case, both values may be located at the different regions in Fig. 2, and $P_{ro}^{b_1}$ result in $q^i = w$ and $q^i_*$ = $w+1$, respectively. If the real remainder lies in the overlapping regions $P_{0,1}^{b_1}$ or $P_{2,1}^{b_1}$, then we should be able to generate $q^i = 1$ and 0 for the truncated and rounded remainder, respectively. This results in both having the same quotient digit, i.e. $q = w+1$. Let $e_{max}$ be the maximum error which causes both truncated and rounded values to be located in the different $q^i$-regions, but they are still in the same overlapping region. For the overlapping region $P_{0,1}^{b_1}$, we have $e_{max} = P_{ro}^{b_1} = P_{1}^{b_1}$ or $P_{ro}^{b_1} = P_{0}^{b_1}$ and $(1/3)D_{max} \leq P_{ro}^{b_1} = P_{0}^{b_1} - e_{max} = c_1 = e_{max}$. Therefore, $e_{max} = c_1 - (1/3) = 0.510_3 - 0.333_3 = 0.171_3 \geq 0.125_3 \geq 0.001_2$, i.e. $P_{ro}^{b_1} = c_1 - e_{max} > 0.0112$. This confirms that the first seven significant digits, $(SP_1, P_0, P_{-1}, P_{-2}, P_{-3}, P_{-4})$, are sufficient in this implementation.