Design of a 16-Bit CMOS Divider/Square-Root Circuit

Hongge Ren, Loc B. Hoang, Hsin-Chia Chen, Belle W. Y. Wei*

Department of Electrical Engineering, College of Engineering, San Jose State University
San Jose, CA 95192-0084, (408) 924-3881, (408) 924-3925 FAX
wei@ee0.sjsu.edu

Abstract

We have developed a 16-bit VLSI circuit for division and square-root operations used extensively in digital signal applications. The circuit uses the nonrestoring method to obtain quotient (root) bits. The quotient (root) value in each iterative step is kept in binary form whereas the partial remainders (radicands) are in redundant binary representations. The iterative core is a redundant binary and binary subtraction, implemented by a carry-save adder. The quotient (root) bit selection logic inputs leading three digits of partial remainders (radicands) and can be implemented in a simple circuit. The resultant circuit in 1.2 μm CMOS technology has an area of 6.72 mm^2 and a speed of 47.7 ns and 49.2 ns for rounded quotient and square-root outputs respectively.

1. Introduction

Division and square-root are important arithmetic operations used extensively in floating point as well as digital signal processors. Consequently, there has been extensive work on developing "fast" algorithms for these difficult, yet basic, operations [1][2][3][4][5][6]. The emphasis of our project is on developing a VLSI implementation for medium width data common in DSP applications [7][8][9].

We have developed a 16-bit shared combinational hardware for computing rounded division and square-root operations. The circuit uses the nonrestoring method to obtain quotient (root) bits. The quotient (root) value in each iterative step is kept in binary form, the result of an on-the-fly redundant binary to binary conversion. The partial remainders (radicands) are in redundant binary representations. The iterative core is a redundant binary and binary subtraction featuring a simple implementation, the same complexity as a carry-save adder. It is a carry-propagation-free subtraction, and its speed is independent of the data width of the subtrahend. The quotient (root) bit selection logic inputs three leading digits of partial remainders (radicands). The resultant modular circuit in 1.2 μm CMOS technology has an area of 6.72 mm^2 and a competitive speed of 47.7 ns [10] for the division operation and 49.2 ns for the square-root operation.

This paper describes our division and square-root algorithms in Sections 2 and 3 respectively. Section 4 discusses the on-the-fly redundant binary to binary conversion required for the square-root operation. Section 5 presents the floor plan and compact circuit layout of our implementation. The last section summarizes our result and discusses how it can be extended.

2. Redundant Binary Division Algorithm

2.1. Division Algorithm

The division algorithm uses the nonrestoring division method. For an n-bit dividend R_0 and divisor D in the range of $[1, 2)$, the quotient bits are determined as follows:

initial step:

$$q_0 = 1,$$

$$R_1 = 2R_0 - 2D,\ R_0 = R_0^t R_0^0 . R_0^1 R_0^2 \ldots R_0^{n-1}$$

where $R_0^t = 0$;

iterative step:

for $1 \le j \le n-1$

$$q_j = \begin{cases} 1 \\ 0 \\ \bar{1} \end{cases}, \quad R_{j+1} = \begin{cases} 2(R_j - D) \\ 2(R_j + 0_{\bar{2}}) \\ 2(R_j + 0) \\ 2(R_j + D) \end{cases}$$

* This work was supported in part by National Science Foundation grant MIP 9019862.

$$\text{if} \quad \begin{cases} (R_j^t R_j^0 . R_j^1)_{BSD} \geq 1 \\ 1 > (R_j^t R_j^0 . R_j^1)_{BSD} \geq 0 \\ 0 > (R_j^t R_j^0 . R_j^1)_{BSD} > -1 \\ (R_j^t R_j^0 . R_j^1)_{BSD} \leq -1 \end{cases}$$

where

$0 \equiv 00.00\ldots0$,

$0_{\bar{2}} \equiv 11.11\ldots1$ with the initial carry-in of 1,

$q_j \in \{\bar{1}, 0, 1\}$ is the jth quotient binary signed digit (BSD),

$(R_j^t R_j^0 . R_j^1)_{BSD}$ is three leading digits of the shifted R_j.

To speed up the addition (subtraction) in updating the partial remainder R_{j+1}, the partial remainder is in redundant binary representation with binary signed digits (BSD). The resultant iterative core is an addition of a redundant binary number to a binary number. Notice that two versions of 0's prevent representation overflow of the redundant R_{j+1} [11].

2.2. Redundant Binary Addition Logic

Redundant binary addition is carry-propagation-free, and its speed is independent of operands' data width. The addition of a redundant binary number to a binary number consists of two steps:

$$a_i + b_i = 2c_i + s_i$$
$$s_i + c_{i+1} = z_i$$

where

$a_i \in \{\bar{1}, 0, 1\}$,

$b_i \in \{0, 1\}$,

$c_i \in \{0, 1\}$, the intermediate carry,

$s_i \in \{0, \bar{1}\}$, the intermediate sum digit,

$z_i \in \{\bar{1}, 0, 1\}$, the sum digit.

These two steps are summarized in Tables 1 and 2.

a_i^l	a_i^r	a_i	b_i	s_i	c_i
0	0	0	0	0	0
0	1	1	0	$\bar{1}$	1
1	0	$\bar{1}$	0	$\bar{1}$	0
1	1	0	0	0	0
0	0	0	1	$\bar{1}$	1
0	1	1	1	0	1
1	0	$\bar{1}$	1	0	0
1	1	0	1	$\bar{1}$	1

Table 1: First Step of Redundant Binary Addition

c_{i+1}	s_i	z_i
0	0	0
0	$\bar{1}$	$\bar{1}$
1	0	1
1	$\bar{1}$	0

Table 2: Second Step of Redundant Binary Addition

The logic for one redundant binary and binary (RB-B) adder can be derived from Table 3 in which $a_i \equiv a_i^l a_i^r$, $z_i \equiv z_i^l z_i^r$, and 00 (or 11), 01, 10 encode 0, 1, $\bar{1}$ respectively:

$$c_i = \overline{a_i^l} a_i^r + b_i \overline{a_i^l} + b_i a_i^r$$
$$z_i^l = a_i^l \oplus a_i^r \oplus b_i$$
$$z_i^r = c_{i+1}$$
$$= \overline{a_{i+1}^l} a^r{}_{i+1} + b_{i+1} \overline{a^l{}_{i+1}} + b_{i+1} a^r{}_{i+1}.$$

a_i^l	a_i^r	b_i	c_{i+1}	z_i^l	z_i^r
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	1	0
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	1	0	1
1	1	0	0	0	0
1	1	0	1	0	1
1	1	1	0	1	0
1	1	1	1	1	1

Table 3: Logic of Redundant Binary Addition

2.3. Quotient Select Logic

Based on $(R_j^t R_j^0 . R_j^1)_{BSD}$ of each step, there are four possible choices for the quotient digit: $1, 0, \bar{0}, \bar{1}$, corresponding to four cases a, b, c, d respectively. This is shown in Table 4. The resultant logic is as follows:

$$SEL1 = \overline{(R_0^l \bar{R}_0^r R_1^l \bar{R}_1^r)}$$

$$SEL2 = \overline{(\bar{R}_0^l R_0^r \bar{R}_1^l R_1^r)}$$

$$a = SEL1(\bar{R}_t^l \cdot R_t^r) + \overline{(R_t^l \oplus R_t^r)}\ (\bar{R}_0^l \cdot R_0^r)(\bar{R}_1^l + R_1^r)$$

$$b = \overline{SEL1}(\bar{R}_t^l \cdot R_t^r) + \overline{(R_t^l \oplus R_t^r)}(\bar{R}_0^l \cdot R_0^r)(R_1^l \cdot \bar{R}_1^r)$$
$$+ \overline{(R_t^l \oplus R_t^r)}\ \overline{(R_0^l \oplus R_0^r)}\ (R_1^l + \bar{R}_1^r)$$

$$c = \overline{(R_t^l \oplus R_t^r)}\ \overline{(R_0^l \oplus R_0^r)}(R_1^l \cdot \bar{R}_1^r) + \overline{SEL2}\ (R_t^l \cdot \bar{R}_t^r)$$
$$+ \overline{(R_t^l \oplus R_t^r)}\ (\bar{R}_0^l \cdot R_0^r)\ (\bar{R}_1^l \cdot R_1^r)$$

$$d = SEL2\ (R_t^l \cdot \bar{R}_t^r) + \overline{(R_t^l \oplus R_t^r)}\ (R_0^l \cdot \bar{R}_0^r)\ (R_1^l + \bar{R}_1^r)$$

where $R^t \equiv R_t^l R_t^r$, $R^0 \equiv R_0^l R_0^r$, $R^1 \equiv R_1^l R_1^r$.

R^t	R^0	R^1	a	b	c	d
0	0	0	0	1	0	0
0	0	1	0	1	0	0
0	0	$\bar{1}$	0	0	1	0
0	1	0	1	0	0	0
0	1	1	1	0	0	0
0	1	$\bar{1}$	0	1	0	0
0	$\bar{1}$	0	0	0	0	1
0	$\bar{1}$	1	0	0	1	0
0	$\bar{1}$	$\bar{1}$	0	0	0	1
$\bar{1}$	0	0	0	0	0	1
$\bar{1}$	0	1	0	0	0	1
$\bar{1}$	0	$\bar{1}$	0	0	0	1
$\bar{1}$	1	0	0	0	0	1
$\bar{1}$	1	1	0	0	1	0
$\bar{1}$	1	$\bar{1}$	0	0	0	1
$\bar{1}$	$\bar{1}$	0	0	0	0	1
$\bar{1}$	$\bar{1}$	1	0	0	0	1
$\bar{1}$	$\bar{1}$	$\bar{1}$	0	0	0	1
1	0	0	1	0	0	0
1	0	1	1	0	0	0
1	0	$\bar{1}$	1	0	0	0
1	1	0	1	0	0	0
1	1	1	1	0	0	0
1	1	$\bar{1}$	1	0	0	0
1	$\bar{1}$	0	1	0	0	0
1	$\bar{1}$	1	1	0	0	0
1	$\bar{1}$	$\bar{1}$	0	1	0	0

Table 4: Quotient Selection Table

3. Square Root Algorithm

The square root operation is similar to division except with a varied divisor during each iterative step:

initial:

$q_0 = 1$

$Q_0 = 01$

$R_1 = R_0 - 1$

where R_0 is the operand of square root;

iterative step:

for $1 \le j \le n-1$

$$q_j = \begin{cases} 1 \\ 0 \\ \bar{1} \end{cases}, \quad R_{j+1} = \begin{cases} 2(R_j - (Q_j + 2^{-j-2})) \\ 2(R_j + 0_{-2}) \\ 2(R_j + 0) \\ 2(R_j + (Q_j - 2^{-j-2})) \end{cases}$$

$$\text{if} \quad \begin{cases} (R_j^t R_j^0 . R_j^1)_{BSD} \ge 1 \\ 1 > (R_j^t R_j^0 . R_j^1)_{BSD} \ge 0 \\ 0 > (R_j^t R_j^0 . R_j^1)_{BSD} > -1 \\ (R_j^t R_j^0 . R_j^1)_{BSD} \le -1 \end{cases}$$

where $Q_j = Q_{j-1} + 2^{-j} q_j$, that is, $Q_j = q_0 q_1 q_2 \cdots q_{j-1}$.

In order to use the RB-B adder specified in the divider, $Q_j + 2^{-j-2}$ or $Q_j - 2^{-j-2}$ has to be in binary form. This is no problem because of on-the-fly conversion from redundant binary Q_j to its binary equivalent.

4. On-the-Fly Redundant Binary to Binary Conversion

The partial redundant binary quotient (root) is converted to its binary equivalent [12] at each iterative step. The on-the-fly conversion minimizes the conversion delay after the full quotient (root) is obtained at the end of iteration. In addition, the iterative core of the square root operation requires a binary partial root as the subtrahend for its redundant binary and binary subtraction.

The conversion algorithm tracks the binary partial quotient Q_j and its binary "minus one" ($Q_j^- = Q_j - 2^{-j}$) in step with the division iteration. The pair (Q_{j+1} and Q_{j+1}^-) is updated based on the value of quotient digit q_{j+1} as follows:

$$Q_{j+1} = \begin{cases} Q_j + 2^{-(j+1)} \\ Q_j \\ Q_j^- + 2^{-(j+1)} \end{cases} \quad \text{if } q_{j+1} = \begin{cases} 1 \\ 0 \\ \bar{1} \end{cases},$$

$$Q_{j+1}^- = \begin{cases} Q_j \\ Q_j^- + 2^{-(j+1)} \\ Q_j^- \end{cases} \quad \text{if } q_{j+1} = \begin{cases} 1 \\ 0 \\ \bar{1} \end{cases}.$$

The initial condition is

$$Q_1 = \begin{cases} 0.1 \\ 0.0 \\ 1.1 \end{cases}, \; Q_1^- = \begin{cases} 0.0 \\ 1.1 \\ 1.0 \end{cases} \quad \text{if } q_1 = \begin{cases} 1 \\ 0 \\ \bar{1} \end{cases}.$$

Let $A_j = Q_j$, $B_j = Q_j^-$, and a_j, b_j, c_j, d_j correspond to q_j being $1, 0, \bar{0}, \bar{1}$ respectively. The logic for $j+1$ leading bits of A_j and B_j for $1 \le j \le 15$ can implemented by a pair of 16-to-1 multiplexers:

$$A_j[t, 0 \ldots j-1] = \begin{cases} A_{j-1}[t, 0 \ldots j-1] \\ B_{j-1}[t, 0 \ldots j-1] \end{cases} \text{if } \begin{cases} d_j \ne 1 \\ d_j = 1 \end{cases},$$

$$B_j[t, 0 \ldots j-1] = \begin{cases} A_{j-1}[t, 0 \ldots j-1] \\ B_{j-1}[t, 0 \ldots j-1] \end{cases} \text{if } \begin{cases} a_j = 1 \\ a_j \ne 1 \end{cases},$$

with the initial $A_0[t, 0]$ and $B_0[t, 0]$ being 01 and 00 respectively. The jth bit for A_j and B_j is produced by a pair of or gates:

$$A_j[j] = a_j[j] + d_j[j]$$

$$B_j[j] = b_j[j] + c_j[j] .$$

5. Implementation

Our 16-it division/square root circuit has four major components: redundant binary adder array, quotient select logic (QSL) circuit, on-the-fly converter, and rounding

circuitry [13]. The QSL circuit and on-the-fly converter are placed in the middle to minimize routing areas, thus splitting the layout into two symmetrical halves and saving 20% chip area as shown in Figure 1.

The modular layout has 17 rows in which the top 16 rows generate pre-rounded quotient (root) bits and the 17th row implements the rounding operation. Each of the top 16 rows consists of 16 redundant binary adder cells, one on-the-fly converter cell, and one quotient select logic cell. The 17th row for rounding has the same elements for generating the 17th quotient (root) bit. In addition, it has circuits for restoring quotient (root) values in case of a negative remainder, and an incrementer for a round-up operation.

The rectangular chip in 1.2 μm CMOS has an area of $2.8mm \times 2.4mm = 6.72mm^2$, and a transistor count of 19,812. The Spice simulated speed is 47.7 ns for the rounded division operation and 49.2 ns for the rounded square-root operation. Eliminating rounding circuitry improves the circuit speed by 20% and area by 8%, whereas eliminating the square-root operation results in a 3.2% speed gain and 5% area savings.

6. Summary

We have developed a compact and regular circuit for division and square-root operations for medium data-width operands common in DSP applications. The regular array structure allows easy insertion of registers for pipeline operations and high throughput.

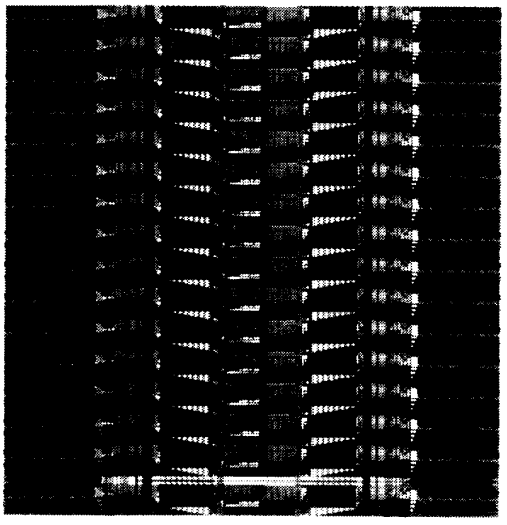

Figure 1: A 16-bit Divider/Square-Root Circuit

References

[1] J. Fandrianto, "Algorithm for High Speed Shared Radix 4 Division and Radix 4 Square-Root," *Proc. 8th IEEE Symposium on Computer Arithmetic, Como, Italy*, pp. 73-79, May 1987.

[2] J. Fandrianto, "Algorithm for High Speed Shared Radix 8 Division and Radix 8 Square-Root," *Proc. 9th IEEE Symposium on Computer Arithmetic, Santa Monica, CA*, pp. 68-75, September 1989.

[3] J.H.P. Zurawski and J.B. Gosling, "Design of a High-Speed Square Root Multiply and Divide Unit," *IEEE Transactions on Computers*, Vol. C-36, No. 1, January 1987.

[4] Daniel E. Atkins, "Higher-Radix Division Using Estimates of the Divisor and Partial Remainders," *IEEE Transactions on Computers*, Vol. C-17, No. 10, October 1968.

[5] Tony M. Carter, "Radix-16 Signed-Digit Division," *IEEE Transactions on Computers*, Vol. 39, No. 12, December 1990.

[6] Stanislaw Majerski, "Square-Rooting Algorithms for High-Speed Digital Circuits," *IEEE Transactions on Computers*, Vol. C-34, No. 8, August 1985.

[7] S. Kuninobu, T. Nishiyama, H. Edamatsu, T. Taniguchi, and N. Takagi, "Design of high speed MOS multiplier and divider using redundant binary representation," in *Proc. 8th Symp. Computer Arithmetic*, pp. 80-86, 1987.

[8] Tamotsu Nishiyama and Shigeo Kuninobu, "Divider and Arithmetic Processing Units Using Signed Digit Operands," *United States Patent*, Patent number 4,935,892, Jun. 19, 1990.

[9] H.R. Srinivas and Keshab K. Parhi, "High-Speed VLSI Arithmetic Processor Architectures Using Hybrid Number Representation," *Journal of VLSI Signed Processing*, April 1992.

[10] Ted E. Williams and Mark A. Horowitz, "A Zero -Overhead Self-Timed 160-ns 54-b CMOS Divider," *IEEE Journal of Solid-State Circuits*, Vol. 26, No. 11, November 1991.

[11] Andre Vandemeulebroecke, Etienne Vanzieleghem, Tony Denayer and Paul G. A. Jespers, "A New Carry-Free Division Algorithm and its Application to a Single-Chip 1024-b RSA Processor," *IEEE Journal of Solid-State Circuits*, Vol. 25, No. 3, pp. 748-755, June 1990.

[12] Milos D. Ercegovac and Tomas Lang, "On-the-Fly Conversion of Redundant into Conventional Representations," *IEEE Trans. on Computers*, Vol. C-36, No. 7, pp. 895-897, July 1987.

[13] Milos D. Ercegovac and Thomas Lang, "Implementation of Module Combining Multiplication, Division, and Square Root," *Proc. of the International Symposium on Circuits and Systems*, pp. 150-153, 1989.