A Cost-Effective Pipelined Divider with a Small Lookup Table

Jong-Chul Jeong, *Student Member*, *IEEE*,
Woo-Chan Park, *Member*, *IEEE*,
Woong Jeong,
Tack-Don Han, *Member*, *IEEE*, and
Moon-Key Lee, *Member*, *IEEE*

Abstract—Current pipelinable dividers require very large lookup tables. We propose a cost-effective pipelinable divider that uses a modified Taylor-series expansion and has a smaller lookup table than other pipelinable dividers. The proposed divider requires about 27 percent less area than the pipelinable divider based on normal Taylor-series expansion in single precision.

Index Terms—Computer arithmetic, division, floating-point, lookup table.

1 Introduction

GENERALLY, dividers have been designed to be small and to use iteration and the latency of division is much longer than that of other arithmetic operations. Division operations have a significant effect on total execution times despite their low frequency of use [1]. However, in view of the growth of VLSI technology and the recent expansion of high-performance applications, dividers without iteration have been proposed. In particular, 3D graphics processing has become an important application of processors so that high-quality pipelinable dividers are required for high-performance 3D graphics processing [2], [3], [4].

The pipelinable-division algorithm based on normal Taylor-series expansion stores the first two terms of the Taylor series in a lookup table, and executes a division by referencing the lookup table in the first step and using a multiplier in the second step [2]. Although this algorithm has a smaller lookup table than other pipelinable-division algorithms, the table for this algorithm is still large.

In this paper, we propose a new modification of the pipelinable-division algorithm based on Taylor-series expansion. According to the 0.25 μm ASIC cell library [5], the latency of the proposed division algorithm is about 44 percent longer, but its area is about 27 percent less than the algorithm of [2] in single precision. Following [6] and [7], we compare the proposed algorithm with other algorithms by using the delays expressed in terms of τ and the area cost estimation expressed in terms of *fa*.

In the remainder of this paper, Section 2 explains related work and Section 3 explains the proposed algorithm and architecture. In Section 4, we calculate the maximum errors so that we can use error analysis to determine the bit-widths of all blocks in the divider. We compare the proposed algorithm with other algorithms in Section 5 and we draw our conclusions in Section 6.

- *J.-C. Jeong and M.-K. Lee are with the Department of Electrical and Electronic Engineering, Yonsei University, 134 shincon-dong, Seodeamun-gu, Seoul 120-749, Korea. E-mail: jongblue@spark.yonsei.ac.kr, mklee@yonsei.ac.kr.*
- *W.-C. Park is with the School of Computer Engineering, Sejong University, 98 Kunja-dong, Kwangjin-gu, Seoul 143-747, Korea. E-mail: pwchan@korea.com.*
- *W. Jeong is with the Diznet Group, Digital Media Research Laboratory, LG Electronics, 16 Woomyoun-dong, Seocho-gu, Seoul 137-724, Korea. E-mail: callisto@lge.com.*
- *T.-D. Han is with the Department of Computer Science, Yonsei University, 134 shinchon-dong, Seodeamun-gu, Seoul 120-749, Korea. E-mail: hantack@kurene.yonsei.ac.kr.*

Manuscript received 30 July 2002; revised 29 Apr. 2003; accepted 31 July 2003.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 117039.

2 Related Work

Division algorithms can be classified into two categories: multiplicative algorithms and subtractive algorithms [8], [9]. A multiplicative algorithm uses hardware integrated with a floating-point multiplier and lookup table and calculates a quotient by approximation. Newton-Raphson and series-expansion algorithms are well-known multiplicative algorithms that calculate a quotient by multiplying a dividend by the reciprocal of the divisor, taken from a lookup table.

An important disadvantage of these algorithms is the large lookup table required for high speed. For example, the lookup table for a 16-bit seed Newton-Raphson or series-expansion divider is 64KB [10]. These 16-bit seed dividers execute a division operation with two iterations in single precision. If 8-bit seed dividers are used, the lookup table size is smaller at 128B, but the latency of these dividers is longer because they require three iterations. An accurate quotient approximation algorithm, which uses Taylor-series expansion, has two lookup tables, requiring 400B for single precision [11].

Hung et al. and Liddicoat and Flynn recently proposed pipelinable-division algorithms. Hung et al.'s division algorithm uses Taylor-series expansion, and its major advantages are a simple architecture and a small lookup table [2]. Liddicoat and Flynn's division algorithm calculates the first three terms of the Newton-Raphson algorithm and reduces latency by using parallelism [3].

A division operation can be represented by Taylor-series expansion as follows [2]:

$$\frac{X}{Y}=\frac{X}{Y_h+Y_l}=\frac{X}{Y_h}\left(1-\frac{Y_l}{Y_h}+\left(\frac{Y_l}{Y_h}\right)^2-\cdots\right). \tag{1}$$

In (1), $Y_h = 2^0y_0 + 2^{-1}y_1 + 2^{-2}y_2 + \cdots + 2^{-(p-1)}y_{p-1}$ and $Y_l = Y - Y_h$. Therefore, $Y_h >> Y_l$ and Y_l/Y_h is approximately zero. Because X and Y are normalized fixed-point numbers, x_0 and y_0 are always one and the variables in (1) have the boundary conditions shown in (2).

$$\begin{aligned} &1 \le X, Y < 2 \\ &1 \le Y_h < 2 - 2^{-(p-1)} \\ &0 \le Y_l < 2^{-(p-1)}. \end{aligned} \tag{2}$$

Equation (3) uses the first two terms of the Taylor-series in (1).

$$\frac{X}{Y} \approx \frac{X(Y_h - Y_l)}{Y_h^2}. \tag{3}$$

According to (3), a division operation is executed by multiplying the dividend X by $Y_h - Y_l$ from the divisor and then multiplying the result by $1/Y_h^2$ from a lookup table. The operation of this division algorithm in single precision is shown in Fig. 1: $Y_h - Y_l$ is computed without subtraction because it is calculated by modifying the Booth encoder of the multiplier that multiplies X by $Y_h - Y_l$ [12].

This division algorithm approximates $1/Y_h^2$ by using a lookup table. The size of this lookup table is decreased by approximation, but is still about 13 KB in single precision. Thus, the divider implemented by this division algorithm has the disadvantage that its area is larger than that of other iterative dividers. Moreover, it is impossible to implement the divider in double precision because its area would be huge. The size of the lookup table is therefore a critical factor to implement a pipelinable divider.

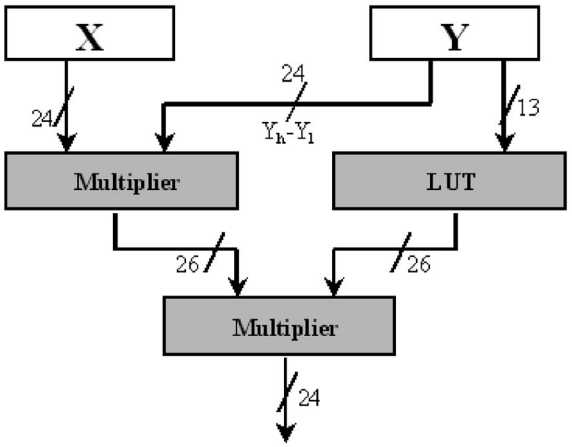

Fig. 1. Block diagram of division algorithm in single precision.

3 Proposed Divider Architecture

The main goal of our proposed algorithm is to reduce the size of the lookup table, the weakest point of the pipelinable divider. First, (3) is executed with a very small lookup table and a coarse quotient is calculated. Second, the subdividend is calculated by multiplying the divisor by this coarse quotient and subtracting the value from the dividend. Next, (3) is executed again, using the subdividend instead of the dividend. Finally, the final quotient is obtained by adding the two calculated quotients. This process seems very complex, but it is executed with a smaller lookup table and uses four multipliers of lower precision than the pipelinable-division algorithm based on normal Taylor-series expansion, by eliminating redundant operations.

3.1 Proposed Algorithm

The proposed algorithm is as follows: The coarse quotient $\tilde{Q}$ is defined as follows by reducing the bit-width of Y_h in (3):

$$\tilde{Q} = \frac{X(Y_h - Y_l)}{Y_h^2}. \tag{4}$$

The subdividend $\tilde{X}$ is calculated by the following equation:

$$\begin{aligned}\tilde{X} &\approx X - Y \cdot \tilde{Q} \\ &= X - Y \cdot \frac{X(Y_h - Y_l)}{Y_h^2} \\ &= X\left(1 - \frac{(Y_h + Y_l)(Y_h - Y_l)}{Y_h^2}\right) \\ &= X\left(1 - \frac{Y_h^2 - Y_l^2}{Y_h^2}\right) = \frac{X \cdot Y_l^2}{Y_h^2}.\end{aligned}$$

Because the dividend X is a positive value, the subdividend $\tilde{X}$ is a positive value. If the dividend X is replaced with the subdividend $\tilde{X}$ in the above equation, the remainder is calculated. The formula is similar, so the remainder is also a positive value.

Now, let us replace X with the subdividend $\tilde{X}$ in (4). If the two quotients are added, the quotient is calculated.

$$\tilde{\tilde{Q}} = \frac{\tilde{X}(Y_h - Y_l)}{Y_h^2}, \qquad \frac{X}{Y} \approx \tilde{Q} + \tilde{\tilde{Q}}.$$

The above equation is rearranged as follows:

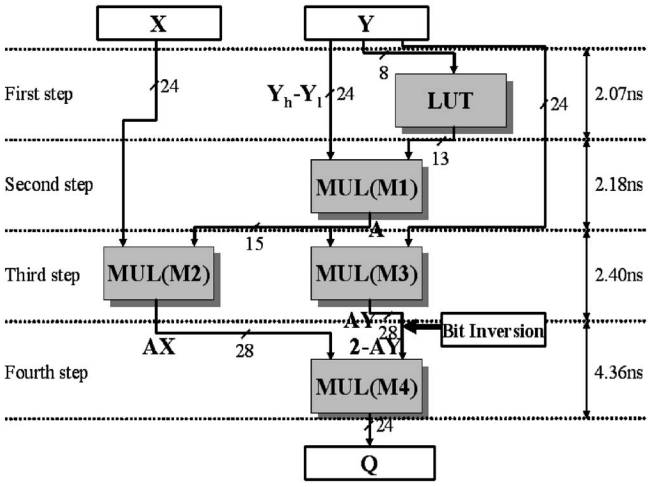

Fig. 2. Block diagram of the proposed algorithm in single precision.

$$\begin{aligned}\frac{X}{Y} &\approx \tilde{Q} + \tilde{\tilde{Q}} \\ &= \frac{(X + \tilde{X})(Y_h - Y_l)}{Y_h^2} \\ &= (X + \tilde{X})A \qquad \text{where } A = \frac{(Y_h - Y_l)}{Y_h^2}. \\ &= (2X - Y\tilde{Q})A \\ &= (2X - AY\ X)A \\ &= (2 - AY)AX,\end{aligned} \tag{5}$$

3.2 Proposed Hardware Architecture

A hardware block diagram for the proposed algorithm is shown in Fig. 2. The execution of (5) is divided into four steps. First, $1/Y_h^2$ is obtained from the lookup table. Second, A is calculated by multiplying $1/Y_h^2$ by $Y_h - Y_l$. Third, AX and AY are calculated through parallel multiplications. Last, the final quotient $AX(2 - AY)$ is calculated by multiplication, where $2 - AY$ can be calculated by the bit-inversion of AY. For example, *0.11010* calculated by the bit-inversion of *1.00101* is smaller by 1 ulp than *0.11011*, the result of *2 – 1.00101*. If bit-inversion is used rather than true subtraction, an adder is removed, but an error is introduced. In Section 4, the influence of this error will be analyzed in detail.

A lookup table and four multipliers are required to execute the proposed algorithm. The latency of this algorithm is 1 LUT + 3 MUL because two multiplications are executed in parallel. The area and latency of three of the four multipliers are somewhat less than those of the pipelinable divider based on normal Taylor-series expansion because their bit-width is smaller. In single precision, the size of the final multiplier is 28×28, two multipliers are 24×15, and the other is 24×13. The value in the lookup table is also accessed in a short time because the size is very small. The number of entries and the bit-width of the lookup table and the bit-width of each multiplier in Fig. 2 will be determined in Section 4 by analyzing errors.

4 Error Analysis

The error analysis of a proposed algorithm is essential for the design of a hardware divider because it provides base data to determine the size of the lookup table and the bit-widths of multipliers. In addition, this algorithm can be compared with other division algorithms by the error analysis. Because the proposed algorithm and the algorithm of [2] permit 1 ulp error, they do not fully support the IEEE floating-point standard [13].

In this paper, we consider four errors caused by the proposed algorithm. The first is the error caused by the restriction in the

number of entries in the lookup table, which is generated by limiting the bit-width of Y_h to p. This is the only error in (5), which assumes that multipliers and the lookup table have infinite precision. Therefore, the first error is calculated by subtracting the result of (5) from the ideal quotient. The second error is that caused by the bit-width restriction of the lookup table, which determines its accuracy. The third error is caused by the rounding positions of the multipliers. The last error is caused by the bit-inversion, which is always 1 ulp.

4.1 Error Due to Entry Restrictions in the Lookup Table

The number of entries in the lookup table is controlled by the bit-width of Y_h. The number of entries is $2^{-(p-1)}$ if the address bit-width in the lookup table is p and Y_h is normalized. The entry restriction error occurs because the number of entries is finite. The error is calculated by subtracting the actual quotient from the ideal quotient, which is calculated by assuming that the bit-widths of X and Y are infinite and that the precisions of the multipliers are infinite.

$$\begin{aligned} E_{Table\text{-}entry} &= \frac{X}{Y} - (2 - AY)AX \\ &= X\left\{\frac{1}{Y} - \frac{Y_h - Y_l}{Y_h^2}\left(2 - \frac{Y_h - Y_l}{Y_h^2}Y\right)\right\} \\ &= X\left\{\frac{1}{Y} - \frac{Y_h - Y_l}{Y_h^2}\left(2 - \frac{Y_h^2 - Y_l^2}{Y_h^2}\right)\right\} \\ &= \frac{X}{Y}\frac{Y_l^4}{Y_h^4}. \end{aligned} \tag{6}$$

By the boundary condition in (2), the maximum error caused by the entry restriction of the lookup table is as follows:

$$\begin{aligned} E_{Table\text{-}entry:MAX} &= \frac{X}{Y}\frac{Y_l^4}{Y_h^4}\Big|_{\substack{X,Y_l=MAX \\ Y,Y_h=MIN}} \\ &< 2 \cdot 2^{-4p+4} \\ &= 2^{-4p+5}. \end{aligned} \tag{7}$$

In addition, (5) can be changed to the following expression using (6):

$$\frac{X}{Y} \approx (2 - AY)AX = \frac{X}{Y}\left(1 - \frac{Y_l^4}{Y_h^4}\right). \tag{8}$$

4.2 Error Due to Bit-Width Restriction in the Lookup Table

The bit-width restriction error is caused because the bit-width of the lookup table is fixed. The first and the fourth of the four errors are always positive. The third error is also positive, because a round-to-zero mode is used in the multipliers to eliminate extra hardware for rounding, as mentioned in Section 4.3. If the second error is made negative, the total error is not increased. When the bit-width of the lookup table is q, the values to round the upper q bits of $1/Y_h^2$ are saved in the lookup table. Thus, a round-to-plus-infinity mode is adopted in this lookup table. The negative value of the error occurs because a round-up is always executed, except that the rounding bit and all sticky bits are zero in a round-to-plus-infinity mode. Thus, the second error is as follows:

$$-2^{-q+1}E_{Table\text{-}bitwidth} \le 0. \tag{9}$$

4.3 Rounding Error in the Multipliers

Four multipliers are used in the proposed algorithm. Rounding is required in each one as, otherwise, the result will have twice the bit-width of the multiplicand in each multiplication, causing fatal problems with regard to area and performance. If round-to-nearest mode is used in a multiplier, the maximum error is reduced, but an adder is required at the output stage of the multiplier. We use round-to-zero mode to eliminate an adder for rounding in each multiplier. In round-to-zero mode, a round-down is always executed and extra hardware for rounding is not required [13]. When the output bit-widths of the multipliers are *m1*, *m2*, *m3*, *m4*, the errors from rounding are as follows:

$$\begin{aligned} 0 \le E_{M1} &< 2^{-m1+1} \\ 0 \le E_{M2} &< 2^{-m2+1} \\ 0 \le E_{M3} &< 2^{-m3+1} \\ 0 \le E_{M4} &< 2^{-m4+1}. \end{aligned} \tag{10}$$

4.4 Bit-Inversion Error

A bit-inversion error of 1 ulp is always generated by replacing the real subtraction with bit-inversion when $2 - AY$ is calculated. This bit-inversion error is dependent on the output bit-width of multiplier *M3*:

$$E_{bit\text{-}inversion} = 2^{-m3+1}. \tag{11}$$

4.5 Total Error

To calculate the total error, it is important to know where each error is generated and how these errors accumulate in this algorithm. The entry restriction error in the lookup table is already included in the term $1/Y_h^2$ in (5). When A is calculated in Fig. 2, the bit-width restriction error in the lookup table and the rounding error in multiplier *M1* are generated. The rounding errors in multiplier *M2* and *M3* are introduced when AX and AY are calculated, respectively. The bit-inversion error is generated when $2 - AY$ is calculated. Finally, the rounding error in multiplier *M4* is generated when $AX(2 - AY)$ is calculated.

The bit-width restriction error in the lookup table and the rounding error in multiplier *M1* are expressed as follows:

$$\begin{aligned} [A]_{include\text{-}error} &= \left(\frac{1}{Y_h^2} - E_{Table\text{-}bitwidth}\right)(Y_h - Y_l) - E_{M1} \\ &= A - E_{Table\text{-}bitwidth}(Y_h - Y_l) - E_{M1}. \end{aligned}$$

Similarly, the other errors are expressed as follows:

$$\begin{aligned} [AX]_{include\text{-}error} &= [A]_{include\text{-}error}X - E_{M2} \\ &= AX - E_{Table\text{-}bitwidth}(Y_h - Y_l)X - E_{M1}X - E_{M2} \\ [AY]_{include\text{-}error} &= [A]_{include\text{-}error}Y - E_{M3} \\ &= AY - E_{Table\text{-}bitwidth}(Y_h - Y_l)Y - E_{M1}Y - E_{M3} \\ [2 - AY]_{include\text{-}error} &= 2 - AY + E_{Table\text{-}bitwidth}(Y_h - Y_l)Y \\ &\quad + E_{M1}Y + E_{M3} - E_{bit\text{-}inversion}. \end{aligned}$$

The quotient including errors is calculated as follows:

$$\begin{aligned} [Q]_{include\text{-}error} = (AX &- E_{Table\text{-}bitwidth}(Y_h - Y_l)X - E_{M1}X - E_{M2}) \times \\ (2 &- AY + E_{Table\text{-}bitwidth}(Y_h - Y_l)Y + E_{M1}Y + E_{M3} \\ &- E_{bit\text{-}inversion}) - E_{M4}. \end{aligned}$$

We can ignore the second-order error terms because the terms in which one error is multiplied by another are very small:

$$\begin{aligned} [Q]_{include\text{-}error} \approx Q &- E_{Table\text{-}entry} + E_{Table\text{-}bitwidth}(AXY(Y_h - Y_l) \\ &- X(2 - AY)(Y_h - Y_l)) \\ &+ E_{M1}(AXY - X(2 - AY)) - E_{M2}(2 - AY) \\ &+ AXE_{M3} - AXE_{bit\text{-}inversion} - E_{M4}. \end{aligned}$$

The above equation is derived by eliminating the second-order error terms and substituting $Q - E_{Table\text{-}entry}$ for $AX(2 - AY)$. The total error can then be found as follows:

TABLE 1
Lookup Table Size in the Proposed Algorithm

m	p	No. of entries	q	ROM size (Bytes)	
15	6	32	8	32	
16	6	32	9	36	
. . .	. . .	. . .	. . .	. . .	
23	7	64	14	112	
24	8	128	13	208	single precision
25	8	128	14	224	
. . .	. . .	. . .	. . .	. . .	
52	15	16 K	27	54 K	
53	15	16 K	28	56 K	double precision

$$E_{Total} \approx E_{Table\text{-}entry} + E_{Table\text{-}bitwidth}(2X(Y_h - Y_l)(1 - AY)) + E_{M1}(2X(1 - AY)) + E_{M2}(2 - AY) - E_{M3}AX + AXE_{bit\text{-}inversion} + E_{M4}.$$

If we substitute $(Y_h^2 - Y_l^2)/Y_h^2$ for AY:

$$E_{Total} \approx E_{Table\text{-}entry} + E_{Table\text{-}bitwidth}2X\frac{Y_l^2}{Y_h^2}(Y_h - Y_l) + E_{M1}\left\{2X\frac{Y_l^2}{Y_h^2}\right\} + E_{M2}\frac{Y_h^2 + Y_l^2}{Y_h^2} - E_{M3}X\frac{Y_h - Y_l}{Y_h^2} + E_{bit\text{-}inversion}X\frac{Y_h - Y_l}{Y_h^2} + E_{M4}. \quad (12)$$

In (12), $E_{Table\text{-}entry}$, $E_{bit\text{-}inversion}$, E_{M1}, E_{M2}, E_{M3}, and E_{M4} are positive errors, while $E_{Table\text{-}bit\text{-}width}$ is a negative error, and the term including E_{M3} is always negative. Therefore, the maximum positive error is determined when $E_{Table\text{-}bit\text{-}width}$ and E_{M3} take their minimum values and the others take maximum values and the maximum negative error is determined in the opposite case. We can derive the following equation by substituting (7), (9), (10), and (11) in (12) and approximating $X = 2$, $Y = 1$, $Y_l^2/Y_h^2 = 2^{-p+2}$:

$$-(2^{-2p-q+5} + 2^{-m3+2})E_{Total}2^{-4p+5} + 2^{-2p-m1+5} + 2^{-p-m2+3} + 2^{-m2+1} + 2^{-m3+2} + 2^{-m4+1}.$$

The maximum permitted limit of total error is determined by the precision of numbers and the accuracy of rounding. To eliminate rounding error, the precise of calculation of numbers must be doubled or an additional operation will be required, as explained in [14]. This is a difficult point in approximation techniques like the multiplicative method. If 1 ulp error is permitted, the maximum permitted limit of total error is as follows:

$$2^{-4p+5} + 2^{-2p-m1+5} + 2^{-p-m2+3} + 2^{-m2+1} + 2^{-m3+2} + 2^{-m4+1}2^{-m+1}; \quad (13)$$

$$2^{-2p-q+5} + 2^{-m3+2}2^{-m+1}. \quad (14)$$

In the above equation, m is the precision of numbers determined by the system specification and p, q, $m1$, $m2$, $m3$, and $m4$ are values that must be determined.

4.6 Selection of the Number of Entries and the Bit-width of the Lookup Table

In (13), p, q, $m1$, $m2$, $m3$, and $m4$are the factors determining the size of the lookup table and multipliers in the proposed divider. p is the number of address bits used to access the lookup table. p is selected first because it is the highest order and most used factor in (13). Ifp increases by one, the number of lookup table entries doubles, so the smallest possible p must be selected. For single precision (m = 24), $p$must be eight according to (13). The value of p for various m is shown in Table 1. The number of entries in the lookup table is $2^{(p-1)}$ because Y is normalized.

After p is selected, q, the bit-width of the lookup table, is selected by (14). The size of multiplier $M1$ is determined by q. q must also be selected with regard to $m3$. Assuming $2^{-2p-q+5} > 2^{-m3+2}$, q is selected by the following equation (if $2^{-2p-q+5} \leq 2^{-m3+2}$, the area of the divider is a little larger than for the present case):

$$2^{-2p-q+5} \leq 2^{-m}.$$

Values of p and q for selected values of m are shown in Table 1.

4.7 Selection of Rounding Positions for Multipliers

$m1$, $m2$, $m3$, and $m4$ represent the rounding positions of the multipliers. In addition, $m1$ affects the sizes of multipliers $M2$ and $M3$. $m2$ and $m3$ affect the size of multiplier $M4$. $m4$ is fixed because it defines the result of the divider. Therefore, $m1$ is selected before $m2$ or $m3$.

$m1$, $m2$, $m3$, and $m4$ are selected as 15, 28, 28, and 25, respectively, when $p = 8$ and $q = 13$ for single precision. Therefore, the size of the multipliers is 24×13, 24×15, 24×15, and 28×28, respectively, as shown in Fig. 2. The number of entries in the lookup table is 128 and the lookup table is 13 bits wide.

5 Comparison with Other Algorithms

In this section, the characteristics and performances of the proposed algorithm are compared with the Newton-Raphson algorithm, the series-expansion algorithm [10], and the accurate quotient approximation algorithm [11]. We also examine the advantages and disadvantages of the proposed algorithm in comparison with the pipelinable-division algorithm based on normal Taylor-series expansion [2].

TABLE 2
Comparison of Pipelineable-Division Algorithms

		The algorithm of [2]	Proposed
General view	Latency	2 MUL or 1 MUL + 1 LUT	1 LUT + 3 MUL
	Throughput	1 cycle	1 cycle
	Number of LUT	1 (no. of entry: $2^{m/2}$, bit-width: $m+2$)	1 (much smaller than the algorithm of [2])
	Number of MUL	2	4
In single precision	Latency (ns)	7.62	11.01
	LUT	13 KB	208 B
	MUL	26×26, 24×24	24×13, 24×15, 24×15, 28×28 or 24×13, 24×17, 24×17, 27×27
In double precision	Latency (ns)	Impossible to implement (LUT > 470 MB)	24.19
	LUT		56 KB
	MUL		53×28, 53×28, 53×28, 58×58

5.1 Comparison with Hung et al.'s Algorithm

The division algorithm of [2] requires one lookup table and two multipliers, as shown in Fig. 1. Its latency is one LUT + one MUL or two MUL. As the accuracy of the divider increases, the latency of the multipliers increases almost linearly according to the 0.25μm ASIC cell library [5], but the latency of the lookup table increases by geometrical progression. The total latency is therefore two MUL at low accuracy and one LUT + one MUL at high accuracy. In single precision, the total latency is two MUL because the latency of the lookup table is less than that of a multiplier. In double precision, the total latency is one LUT + one MUL because the latency of the lookup table is longer.

The latency of the proposed algorithm is one LUT + three MUL, but the lookup table is smaller than that of the divider of [2] and two out of three multipliers have shorter latencies because of their small bit-width. The latency is thus a little longer than the algorithm of [2] in single precision.

In single precision, the latency of the proposed algorithm is 11.01 ns, whereas that of the algorithm of [2] is 7.62 ns, when simulated with 0.25μm ASIC cell library [5]. The latency of the proposed algorithm is 24.19 ns in double precision, but it is impossible to implement the algorithm of [2] in double precision because of its huge lookup table. The major features of the two algorithms are summarized in Table 2. The areas for the two algorithms based on the 0.25μm ASIC cell library are shown in Fig. 3. On this basis, the proposed algorithm requires 27 percent less area than the algorithm of [2]. The proposed algorithm is a cost-effective pipelinable division that has advantages with regard to chip area.

5.2 Comparison with Other Algorithms

The comparison with other algorithms is very important but very difficult. Following [6] and [7], the delays are expressed in terms of τ—the delay of a complex gate, such as one full adder, and the unit employed for the area cost estimation is the size of one full adder, *fa*. These delay and area cost estimations depend on the technology employed and on the actual implementation, but, because all the schemes use similar logic blocks, the relative values may express a general trend among the different designs. The comparison with other algorithms in single precision is shown in Table 3 and the comparison in double precision is shown in Table 4. The delay and the area cost estimation of the proposed algorithm and the division algorithm of [2] are fully explained in the next paragraph. The function units of Newton-Raphson, series-expansion and accurate quotient approximation algorithms are shown in the tables of [10] and [11].

In single precision, the delay and the area cost of the proposed algorithm are as follows:

$$\begin{aligned}\text{Delay} &: 4\tau\ (\text{table}) + 7.5\tau(24\times 13\ \text{MUL}) + 8\tau(24\times 15\ \text{MUL}) \\ &\quad + 9.5\tau(28\times 28\ \text{MUL}) = 29\tau \\ \text{Area cost} &: \text{Table}\ (2^7\times 13) + 24\times 13\ \text{MUL} + (2\times)24\times 15\ \text{MUL} \\ &\quad + 28\times 28\ \text{MUL} \\ &= 57fa + 345fa + 2\times 418fa + 972fa = 2{,}210fa.\end{aligned}$$

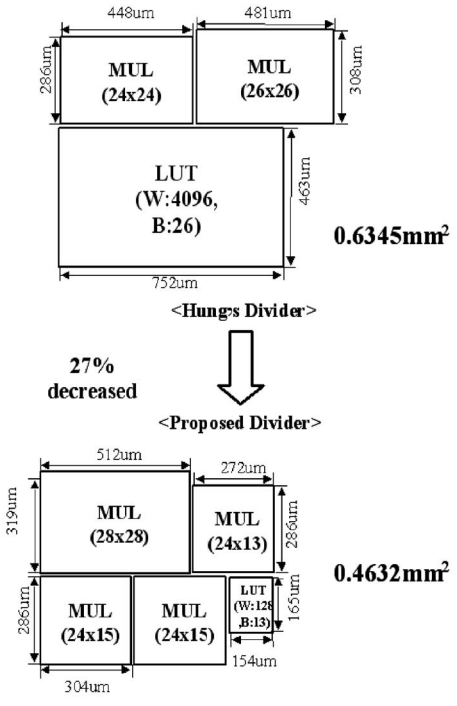

Fig. 3. Area comparison of pipelinable-division algorithms.

TABLE 3
Comparison with Other Algorithms in Single Precision

Algorithm and implementation method		Latency (τ)	Pipelin -ability	Accurate remainder	No. of iterations	Area (fa)	No. of MULs	Size of LUT
Newton-Raphson	8 b seed	51.5	×	×	3	912	1	128 B
	16 b seed	36.5	×	×	2	13,677	1	64 KB
Series-expansion	8 b seed	61	×	×	3	912	1	128 B
	16 b seed	46	×	×	2	13,677	1	64 KB
Accurate quotient approximation		39	×	○	2	1,591	3	400 B
The algorithm of [2]		18.5	○	×	1	5,301	2	13 KB
Proposed		29	○	×	1	2,210	4	240 B

TABLE 4
Comparison with Other Algorithms in Double Precision

Algorithm and implementation method		Latency (τ)	Pipelin -ability	Accurate remainder	No. of iterations	Area (fa)	No. of MULs	Size of LUT
Newton-Raphson	8 b seed	51.5	×	×	3	912	1	128 B
	16 b seed	36.5	×	×	2	13,677	1	64 KB
Series-expansion	8 b seed	61	×	×	3	912	1	128 B
	16 b seed	46	×	×	2	13,677	1	64 KB
Accurate quotient approximation		39	×	○	2	1,591	3	400 B
The algorithm of [2]		18.5	○	×	1	5,301	2	13 KB
Proposed		29	○	×	1	2,210	4	240 B

In double precision, the delay and the area cost of the proposed algorithm are as follows:

$$\begin{aligned}&\text{Delay}: 8\tau\ (\text{table}) + 2 \times 10.5\tau(53 \times 28\ \text{MUL}) + 12\tau(58 \times 58\ \text{MUL})\\ &\quad = 41\tau\\ &\text{Area cost}: \text{Table};\ (2^{14} \times 28) + (3\times)53 \times 28\ \text{MUL} + 58 \times 58\text{MUL}\\ &= 11,200fa + 3 \times 1,221fa + 2,712fa = 17,575fa.\end{aligned}$$

On the other hand, the delay and the area cost of the division algorithm of [2] in single precision are as follows:

$$\begin{aligned}&\text{Delay}: 9\tau(24 \times 24\ \text{MUL}) + 9.5\tau(26 \times 26\ \text{MUL}) = 18.5\tau\\ &\text{Area cost}: \text{Table}(2^{12} \times 26) + 24 \times 24\ \text{MUL} + 26 \times 26\ \text{MUL}\\ &= 3,640fa + 784fa + 877fa = 5,301fa.\end{aligned}$$

The delay and the area cost of the division algorithm of [2] are not provided in double precision because it requires a huge (about 470 MB) lookup table.

The proposed algorithm has a larger area cost than the 8-bit seed Newton-Raphson, 8-bit seed series-expansion, and accurate quotient approximation algorithms, but has a shorter latency and is pipelinable. Therefore, the proposed algorithm can be used efficiently in systems where the divider is used very frequently.

6 Conclusion

We propose a new pipelinable-division algorithm without iteration and compare the proposed algorithm with other algorithms. The proposed algorithm requires one lookup table and four multipliers and requires less area than other algorithms in single precision. The proposed divider can be used efficiently in systems where the divider is used very frequently, such as 3D graphics accelerators [4]. We performed an error analysis on the proposed divider and simulated it using Verilog HDL.

References

[1] S.F. Oberman and M.J. Flynn, "Design Issues in Division and Other Floating Point Operations," *IEEE Trans. Computers*, vol. 46, no. 2, pp. 154-161, Feb. 1997.
[2] P. Hung, H. Fahmy, O. Mencer, and M.J. Flynn, "Fast Division Algorithm with a Small Lookup Table," *Conf. Record 33rd Asilomar Conf. Signals, Systems, and Computers*, vol. 2, pp. 1465-1468, May 1999.
[3] A.A. Liddicoat and M.J. Flynn, "Pipelinable Division Unit," Technical Report No. CSL-TR-00-809, Computer Systems Laboratory, Stanford Univ., pp. 11-28, Sept. 2000.
[4] A. Kugler, "The Setup for Triangle Rasterization," *Proc. 11th Eurographics Workshop Computer Graphics Hardware*, pp. 49-58, Aug. 1996.
[5] Samsung Electronics Co. Ltd., *MDL110 0.25 μm 2.5 V CMOS Standard Cell Library for Pure Logic/MDL Products*, 1999.
[6] J.A. Pineiro and J.D. Bruguera, "High-Speed Double-Precision Computation of Reciprocal, Division, Square Root, and Inverse Square Root," *IEEE Trans. Computers*, vol. 51, no. 12, pp. 1377-1388, Dec. 2002.

[7] J.A. Pineiro and J.D. Bruguera, "High-Speed Double-Precision Computation of Reciprocal, Division, Square Root, and Inverse Square Root," technical report, http://www.ac.usc.es, 2001.
[8] I. Koren, *Computer Arithmetic Algorithms.* Prentice Hall, 1993.
[9] D. Ercegovac and T. Lang, *Digital Systems and Hardware/Firmware Algorithms.* John Wiley & Sons, 1985.
[10] S.F. Oberman and M.J. Flynn, "Division Algorithms and Implementations," *IEEE Trans. Computers,* vol. 46, no. 8, pp. 833-854, Aug. 1997.
[11] D. Wong and M.J. Flynn, "Fast Division Using Accurate Quotient Approximations to Reduce the Number of Iterations," *IEEE Trans. Computers,* vol. 41, no. 8, pp. 981-985, Aug. 1992.
[12] C.N. Lyu and D. Matula, "Redundant Binary Booth Recoding," *Proc. IEEE Symp. Computer Arithmetic,* pp. 50-57, July 1995.
[13] ANSI/IEEE Standard 754-1985, *IEEE Standard for Binary Floating-Point Arithmetic,* 1985.
[14] P. Soderquist and M. Leeser, "Division and Square Root: Choosing the Right Implementations," *IEEE Micro,* vol. 17, pp. 56-66, July/Aug. 1997.