A Latency-Effective Pipelined Divider for Double-Precision Floating-Point Numbers

JUWON YUN1, JINYOUNG LEE1, WOO-NAM CHUNG1, CHEONG GHIL KIM2, (Member, IEEE), AND WOO-CHAN PARK1

1Department of Computer Engineering, Sejong University, Seoul 05006, South Korea
2Department of Computer Science, Namseoul University, Cheonan 31020, South Korea
Corresponding author: Woo-Chan Park (pwchan@sejong.ac.kr)

This work was supported in part by the Institute of Information and communications Technology Planning and Evaluation (IITP) grant funded by the Korean Government (MSIT) (Development of Mobile GPU Hardware for Photo-Realistic Realtime Virtual Reality) under Grant 2016-0-00204, and in part by the National Research Foundation of Korea (NRF) grant funded by the Korean Government (MSIT) under Grant 2019R1A2C1005163.

ABSTRACT In this article, we propose an effective algorithm of pipelined dividers for double-precision floating-point numbers. This reduces the latency of the previous pipelined dividers without increasing the lookup table size. The experimental results show that the proposed divider reduces the overall latency up to 16% compared to the previous divider.

INDEX TERMS Computer arithmetic, division, floating-point, lookup table.

I. INTRODUCTION

Generally, the latency of division is much longer than for other arithmetic operations. Division operations have a significant effect on total execution times, despite their low frequency of use [1]. Thus, the design of high-performance dividers has become an important issue in high-speed computing. Efficient pipelined dividers are required for 3D computer graphics, signal processing, and scientific computing applications where the performance depends heavily on division operations [2]–[6]. This efficient pipeline divider technology is widely used to reduce power consumption and latency in multi-media and signal processing applications as well as fast processing [7].

In modern microprocessors, improving performance by reducing the pipeline latency becomes more difficult than by increasing the hardware size. The main reason for the recent trend of using semiconductor technology is that the hardware size increases significantly, but the clock speed is congested. Moreover, reducing pipeline latency can lead to the removal of several hardware components. For example, if a particular application-specific processor, such as GPU, requires a large number of bits of the register per pipeline, we can remove these bits of the register by reducing the pipeline latency.

In addition, the decision of the pipeline latency may affect the overall architecture, so that it may be necessary to fix the pipeline latency of the specific unit. For example, all 3DNow! Instruction in [8] has a fixed two-cycle latency to simplify the overall architecture. A high-radix pipelinable division algorithm based on Taylor series expansion was proposed in [9] and took the first two terms of the Taylor series. Although this algorithm had a smaller lookup table (LUT) than other approaches, the table for this algorithm was still large. To reduce the LUT size significantly, a modification to this pipelinable division algorithm was introduced in [4]. Compared to [9], this algorithm reduces the LUT size from 13 KB to 208 B for single-precision and from 470 MB to 56 KB for double-precision. A further modification for double-precision floating-point numbers was proposed in [5]; this significantly reduces the chip area in comparison to [4] mainly by reducing the LUT size from 56 KB to 2.5 KB. Similar to [5], [10] took the first six terms of Taylor series expansion for the approximation to reduce the size of the LUT.

The size of the LUT and the computational latency are the most important elements in the design of the high-performance divider. The divider architecture in [9] guarantees low latency with a relatively larger LUT. For the double-precision in particular, [9] requires a LUT of about 470 MB, which is impractically large. In the case of [4] and [5], the LUT size can be sufficiently reduced to...
implement double-precision with increased computational latency.

In this article, a novel high-performance divider architecture geared toward reducing computational latency with a similar LUT size to [4] and [5] is proposed. In [4] and [5], the multiplication step is performed after accessing the LUT, while in the proposed architecture, the LUT access and multiplication are processed in parallel using additional multipliers. In this way, the proposed architecture reduces the pipeline depth by one step compared to previous schemes. As a result, this can be expected to be the highest-speed double precision pipelined divider with a reasonable area among the high-performance divider introduced so far. It can definitely give users more choices to select the performance optimization option in selecting the divider.

Because all pipelinable division algorithms permit 1ulp (acronym for unit in the last place) error, they do not fully support the IEEE floating-point standard [12]. This means that our pipelined divider does not perform the rounding and the remainder needs to be calculated.

We calculated the type of error that can occur in order to consider the accuracy required for double-precision in the proposed architecture. Through calculation, the optimal block size considering accuracy is obtained and applied to the proposed architecture. To verify the proposed hardware divider, we compared the proposed hardware divider with other pipelinable divider algorithms and traditional multiplication algorithms.

Similar to [4] and [5], we compare the proposed algorithm with other algorithms by using the delays and the area cost estimation. According to the comparison results, the latency of the proposed division algorithm is about 16% and 7% faster, with an area of about 21% and 45% larger than the algorithms in [4] and [5], respectively.

In the remainder of this article, Section 2 explains 2 related work and Section 3 explains the proposed algorithm and architecture. In Section 4, we perform an error analysis to determine the bit-widths of all blocks in the divider. We compare the proposed algorithm with others in Section 5, and conclusions are provided in Section 6.

II. RELATED WORK

In [9], Y is decomposed into two groups: the higher order bits (Yh) and the lower order bits (Yl). The operands X and Y are normalized m bit fixed-point numbers. Yh is the higher order p bits of Y and Yl is the lower (m-p) bits of Y. That is, $Y_h = 2^p y_0 + \cdots + 2^{-1} y_1 + \cdots + 2^{-m} y_{m-1}$, and $Y_l = Y - Y_h$. A division operation can be represented as follows:

$$\frac{X}{Y} = \frac{X}{Y_h + Y_l} = \frac{X(Y_h - Y_l)}{Y_h^2 - Y_l^2}. \quad (1)$$

The variables in (1) are bounded as follows:

1. $1 \leq X$, $Y < 2$,
2. $1 \leq Y_h < 2^{-1(p-1)}$,
3. $0 \leq Y_l < 2^{-1(p-1)}$.

Using Taylor series, (1) can be expanded at $Y_l/Y_h$ as follows:

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

Hung’s algorithm in [9] combines the first two terms of the Taylor series for approximation, as shown in the next equation.

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

Hung’s algorithm multiplies the dividend X by $(Y_h - Y_l)$ from the divisor and then multiplies the result by $1/Y_h^2$ from a LUT. As mentioned in [4] and [9], $(Y_h - Y_l)$ is computed without subtraction because it is calculated by modifying the Booth encoder of the multiplier [11]. The size of the LUT is estimated to be about 13 KB for single-precision and 470 MB for double-precision. Although it is a very simple architecture, it still requires a large (or even huge for double-precision) LUT.

Jeong’s algorithm in [4] reduces the size of the LUT significantly. It determines a coarse quotient $\hat{Q}$ by executing (2) with a very small LUT; then the sub dividend $\tilde{X}$ is calculated by multiplying the divisor by this coarse quotient and subtracting the value from the dividend: $\tilde{X} = X - Y \hat{Q}$. With this sub dividend, $\tilde{Q}$ is obtained again using (2). The final quotient is obtained by adding the two calculated quotients. This algorithm can be summarized as follows:

$$\frac{X}{Y} \approx \hat{Q} + \tilde{Q} = AX + A\tilde{X} = A(X + \tilde{X})$$

$$= (X + X - Y \hat{Q})A = (2X - YAX)A = (2 - AX)AX. \quad (3)$$

where $A = (Y_h - Y_l)/Y_h^2$.

The hardware architecture of Jeong’s algorithm is shown in Figure 1. It consists of a LUT and four multipliers (MULs)
and requires four steps to complete its pipeline. The size of LUT is estimated to be about 208 B for single-precision and 56 KB for double-precision.

To reduce the size of LUT in [4] for double-precision, Singh’s algorithm in [5] calculates $\tilde{Q}$ as well as $\tilde{Q}$ and $\hat{Q}$. This algorithm can be represented as follows:

$$\frac{X}{Y} \approx \hat{Q} + \tilde{Q} + \tilde{Q} = \left( (2 - AY)^2 - (1 - AY) \right) AX. \quad (4)$$

where $(\hat{Q} + \tilde{Q}) = AX(2 - AY)$ as (3) and $\tilde{Q} = A(X - Y(\hat{Q} + \tilde{Q})) = A(X - Y(AX + AX(1 - AY)))$.

Equation (3) of Jeong’s algorithm can be rewritten as follows in order to exploit the parallelism:

$$\frac{X}{Y} \approx (2 - AY)AX = \frac{1}{Y^2_h}(Y_h - Y_j)X. \quad (5)$$

FIGURE 2. Block diagram for Singh’s algorithm in double-precision.

The hardware architecture of Singh’s algorithm is shown in Figure 2. It consists of one LUT, five MULs, and a carry propagation adder, requiring five steps to complete its pipeline. Note that the multiplication and addition in the fourth step can be implemented by one multiply-and-add (MAC) operator. The size of LUT for double-precision is estimated to be about 2.5 KB. This algorithm reduces the chip area by about 81% compared to the proposed divider in [4].

III. PROPOSED DIVIDER ARCHITECTURE

To propose a latency-effective pipelined hardware divider, we did the following. First, we performed error analysis on the proposed divider architecture, case 1 and case 2, to get the basic data for determining the LUT size and MUL bit width. Second, we calculated the optimal block size by calculating the LUT size and MUL bit-width determined from the basic data. Finally, the divider architecture was proposed through the application of the optimal block size.

According to Figure 1 and Figure 2, the value A is produced in the second step after the LUT is accessed in the first step. The two operations are processed sequentially, which increases the total computational latency. The proposed architecture parallelizes the LUT access and multiplication to reduce the total latency.

In (5), the LUT access for $1/Y^2_h$ and multiplications $(Y_h - Y_j)X$ and $(Y_h - Y_j)Y$ can be parallelized. Figure 3 illustrates the procedural steps based on the (5). The latency of Jeong’s algorithm is 1 LUT + 3 MULs, while the latency of the proposed algorithm is 3 MULs. The proposed algorithm requires one more multiplier compared to Jeong’s algorithm.

Equation (4), based on Singh’s algorithm, can be rewritten as follows to exploit the parallelism:

$$\frac{X}{Y} \approx ((2 - AY)^2 - (1 - AY))AX = ((2 - \frac{1}{Y^2_h}(Y_h - Y_j))^2 - (1 - \frac{1}{Y^2_h}(Y_h - Y_j))Y) \frac{1}{Y^2_h}(Y_h - Y_j)X. \quad (6)$$

The LUT access for $1/Y^2_h$ and the multiplications $(Y_h - Y_j)X$ and $(Y_h - Y_j)Y$ can be performed in parallel. The proposed procedural steps to exploit the parallelism are depicted in Figure 4. The latency of Singh’s algorithm is 1 LUT + 4 MULs, while the latency of the proposed algorithm is 4 MULs with one more multiplier.

IV. ERROR ANALYSIS

The error analysis of the proposed algorithm is essential for the design of the hardware divider, because it provides base data to determine the size of the LUT and the bit-widths of the MULs. In this section, error analysis for the two cases is carried out and then the optimal size of each block is obtained.

A. ERROR DUE FOR CASE 1

There are four types of errors associated with Figure 3. The first error is caused by the restriction in the number of entries in the LUT, which is generated by limiting the bit-width of $Y_h$ to $p$. This error can be calculated by subtracting the result of (5) from the ideal quotient. The second error is
caused by the bit-width restriction of the LUT, which determines its accuracy. The third error is caused by the rounding positions of the MULs. Finally, the last error is caused by the bit-inversion when (2-AY) is calculated, which is always 1 ulp. Note that the first, second, and fourth errors are the same as those of [4].

1) FOUR TYPES OF ERRORS
The first error of the proposed method is the same as that of [4], and thus it is represented as follows:

\[ E_{TE} = \frac{X}{Y} - AX(2-AY) = \frac{X}{Y} \times \frac{Y^4}{Y^h}, \]

\[ E_{TE,\text{MAX}} = \frac{X}{Y} \times \frac{Y^4}{Y^h} < 2 \times 2^{-4p+4} = 2^{-4p+5}. \]

The table-entry error \( E_{TE} \) is obtained 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 MULs are infinite.

When the bit-width of the LUT is \( q \), the values to round the upper q bits of \( 1/Y^2 \) are stored in the LUT. Thus, the second error \( E_{TB} \) is as follows:

\[ 2^{(q+1)} < E_{TB} \leq 0. \]

There are five MULs in Figure 3. As in [4], the round-to-zero mode is used to eliminate an adder for rounding in each MUL. When the output bit-widths of the MULs are m1, m2, m3, m4, and m5, the rounding errors are as follows:

\[ 0 \leq E_{M1} < 2^{-m1+1}, \quad 0 \leq E_{M2} < 2^{-m2+1}, \]

\[ 0 \leq E_{M3} < 2^{-m3+1}, \quad 0 \leq E_{M4} < 2^{-m4+1}, \quad 0 \leq E_{M5} < 2^{-m5+1}. \]

An error of 1 ulp is generated when (2-AY) is calculated. This bit-inversion error \( E_{BI} \) is dependent on the output bit-width of multiplier M4. Thus, \( E_{BI} \) is as follows:

\[ E_{BI} = 2^{-m4+1}. \]

2) TOTAL ERROR
To calculate the total error, it is important to know how each error is propagated. In the first step, three operations of the LUT access for \( 1/Y^2 \), and multiplications \((Y_h-Y_l)X\) and \((Y_h-Y_l)Y\) are accomplished in parallel. The bit-width restriction error in the LUT, the rounding error in M1, and the rounding error in M2 are generated for each operation. Thus the result of each operation including error (IE) is as follows:

\[ [AX]_E = (X(Y_h-Y_l)-E_{M1}) \times \frac{1}{Y^h} - E_{M3}. \]

\[ AX-E_{TB}(Y_h-Y_l)X - E_{M1} \frac{1}{Y^h} - E_{M3}. \]

\[ [AY]_E = (Y(Y_h-Y_l)-E_{M2}) \times \frac{1}{Y^h} - E_{M4}. \]

In the second step, two multiplication operations to calculate \( AX \) and \( AY \) are performed, so that the results for each operation including errors are as follows:

\[ [2-AY]_E = 2-AY+E_{TB}Y(Y_h-Y_l)+E_{M2} \frac{1}{Y^h} + E_{M4} - E_{BI}. \]

According to (5), the final quotient \( Q \) is calculated by AX and (2-AY) in multiplier M5. The final quotient including errors is calculated as follows:

\[ [Q]_E = (AX - E_{TB}(Y_h-Y_l)X - E_{M1} \frac{1}{Y^h} - E_{M3}) \times \left( 2-AY+E_{TB}Y(Y_h-Y_l)+E_{M2} \frac{1}{Y^h} + E_{M4} - E_{BI} \right) - E_{M5}. \]

The second-order error terms can be ignored because the terms are very small. The total error \( E_{\text{total}} \) can then be calculated as follows:

\[ E_{\text{total}} = E_{TE} + E_{TB}2X(Y_h-Y_l) \frac{Y_l}{Y_h} + E_{M1} \frac{1}{Y^h} (1+Y^2_h). \]

In the above equation, \( E_{TE}, E_{M1}, E_{M3}, E_{M5}, \) and \( E_{BI} \) are positive errors, while \( E_{TB} \) is a negative error and the term
including $E_{M2}$ and $E_{M4}$ are always negative. Thus the maximum positive error is determined when $E_{TB}$, $E_{M2}$, and $E_{M4}$ take their minimum values and the others take maximum values. The maximum negative error is determined in the opposite case. According to the case in [4], we can derive the following equation by approximating $X$:

$$E_{\text{total}} < 2^{-4p} + 2^{-m1+1} + 2^{-2p-m1+3} + 2^{-m3+1} + 2^{-2p-m3+3} + 2^{-m5+1} + 2^{-m4+2} - 2^{-2p-q+5} - 2^{-m2+2} - 2^{-m4+2}.$$ 

Thus, the maximum positive error and maximum negative error are as follows:

$$E_{\text{max, pos}} = 2^{-4p} + 2^{-m1+1} + 2^{-2p-m1+3} + 2^{-m3+1} + 2^{-2p-m3+3} + 2^{-m5+1} + 2^{-m4+2} - 2^{-2p-q+5} - 2^{-m2+2} - 2^{-m4+2},$$

$$E_{\text{max, neg}} = 2^{-2p-q+5} + 2^{-m2+2} + 2^{-m4+2}.$$ 

According to the selection processes of [4], we can obtain $p = 15$, $q = 28$, $m1 = 56$, $m2 = 56$, $m3 = 56$, $m4 = 57$, and $m5 = 53$ for double-precision.

### V. COMPARISON WITH OTHER ALGORITHMS

In this section, the characteristics and performances of the proposed algorithm are compared with respect to the previous pipelinable division algorithms and the traditional multiplicative algorithms. The delay and area cost have been calculated based on the analytical method in [13] and [14]. The delays are expressed in terms of $\tau$, which is the delay of a complex gate such as one full adder. The unit employed for the area cost estimation is the size of one full adder, $fa$.

#### A. COMPARISON WITH PREVIOUS PIPELINABLE DIVISION ALGORITHMS

The comparison, in terms of the delay and area cost of the proposed scheme in relation to previous pipelinable division algorithms, is provided in Table 1. Total delay (or total area cost) for each algorithm is calculated by adding the delay (or area cost) of each pipeline step.

According to the results in Table 1, the proposed algorithm (Case 1) could reduce the critical path delay by about 16%, with a 21% larger hardware area compared to [4]. The proposed algorithm (Case 2) could reduce the critical path delay by about 7%, with a 45% larger hardware area compared to [5].

The proposed algorithm (Case 1) could reduce the pipeline depth by one step compared to [4], while the proposed algorithm (Case 1) and [4] have the same value for the longest delay (12.0$\tau$) within a pipeline. This result means that the proposed algorithm (Case 1) can improve the pipeline latency by 25% compared to [4] with the same clock frequency. This result can be applied to the case of the proposed algorithm (Case 2), which can improve the pipeline latency by 20% compared to [5] with the same clock frequency. Compared to [4], the proposed algorithm (Case 2) has the same pipeline depth as [4] and could reduce the area cost by 28% with an 8% slower clock frequency.

### TABLE 1. Estimation results through delay ($\tau$) and area cost ($fa$) Comparison with [4], [5].

|                | Jeong's algorithm [4] | Proposed Case 1 | Singh's algorithm [5] | Proposed Case 2 |
|----------------|-----------------------|------------------|-----------------------|-----------------|
|                | Delay ($\tau$)        | Area ($fa$)     | Delay ($\tau$)        | Area ($fa$)     | Delay ($\tau$) | Area ($fa$) |
| 1st step       | 56KB: 8.0             | 56KB: 11,200    | 53*53: 12.0           | 53*53*2: 4,700  | 2.5KB: 5.7     | 700         |
| 2nd step       | 53*28: 10.5           | 53*29: 10.5     | 56*29: 2,690          | 53*20: 10.0     | 57*19: 10.0   | 826         |
| 3rd step       | 53*28: 10.5           | 53*28*2: 10.5   | 56*57: 2,621          | 53*20: 10.0     | 57*58(MAC): 58*58(MAC): 13.0 |
| 4th step       | 58*58: 12.0           | 58*58: 2,764    | -                     | -               | 58*58(MAC): 12.0 |
| 5th step       | -                     | -                | -                     | -               | 2,794         |
| Total          | 41.0                  | 17,519          | 34.5                  | 21,211          | 50.7          | 8,736       |

#### B. ERROR DUE FOR CASE 2

The final formula for the maximum positive error and maximum negative error are as follows:

$$E_{\text{max, pos}} = 2^{-6p+7} + 6 * 2^{-4p-q+5} + 2^{-m1+2} + 2^{-m3+2} + 2^{-m4+3} + 2^{-m5+2} + 2^{-m6+1},$$

$$E_{\text{max, neg}} = 2^{-m2+2} + 2^{-m4+2}.$$ 

Considering the accuracy requirement for double-precision, we can obtain $p = 11$, $q = 19$, $m1 = 57$, $m2 = 55$, $m3 = 57$, $m4 = 58$, $m5 = 57$, and $m6 = 53$. 

VOLUME 8, 2020
TABLE 2. Actual synthesis results through Delay (n second) and Area (µm²) Comparison with [4], [5].

|                | Jeong’s algorithm [4]          | Proposed Case 1                     | Singh’s algorithm [5]          | Proposed Case 2                     |
|----------------|--------------------------------|-------------------------------------|--------------------------------|-------------------------------------|
|                | Delay (n second) | Area (µm²) | Delay (n second) | Area (µm²) | Delay (n second) | Area (µm²) | Delay (n second) | Area (µm²) |
| 1st step       | 56KB:             | 56KB:     | 53*53:          | 56KB: 24738,   | 2.5KB:             | 2.5KB:     | 53*53:          | 2.5KB: 2257,   |
|                | 1.34              | 24,738    | 1.5             | 53*53*2:       | 1.24              | 2257       | 1.5             | 53*53*2:       |
| 2nd step       | 53*28:           | 53*28:    | 56*29:          | 56*29*2:       | 53*20:            | 53*20:     | 57*19:          | 57*19,55*19:  |
|                | 1.45              | 5,025     | 1.45            | 11,029         | 1.25              | 4,588      | 1.4             | 7,593         |
| 3rd step       | 53*28:           | 53*28*2:  | 56*57:          | 56*57*2:       | 53*20:            | 53*20*2:   | 58*58(MAC):     | 58*58(MAC):   |
|                | 1.45              | 10,076    | 1.5             | 13,607         | 1.28              | 9,306      | 1               | 12,063        |
| 4th step       | 58*58:           | 58*58:    | -               | -               | 58*58(MAC):       | 58*58(MAC):| 57*57:          | 57*57:        |
|                | 1.5              | 13,256    | -               | -               | 1.9               | 12,063     | 1.55            | 11,333        |
| 5th step       | -                | -         | -               | -               | 58*58:            | 58*58:     | 1.5             | 13,256        |
| Total          | 5.74             | 53,095    | 4.45            | 71,499         | 7.17              | 41,470     | 6.35            | 55,371        |

TABLE 3. Comparison with traditional multiplicative algorithms.

| Algorithm and implementation method | Newton-Raphson 8-b seed | Series-expansion 16-b seed | Accurate-quotient approximation 8-b seed | Proposed Case 1 | Proposed Case 2 |
|-------------------------------------|--------------------------|----------------------------|------------------------------------------|-----------------|-----------------|
| Latency (n)                         | 88                       | 68                         | 100                                      | 80              |                 |
| Pipelibility                        | ×                        | ×                          | ×                                        | ×               | ×               |
| Accuracy                           | ×                        | ×                          | ×                                        | ×               | ×               |
| No. of iterations                   | 4                        | 3                          | 4                                        | 3               | 1               |
| Area (µm²)                          | 2,635                    | 15,400                     | 2,635                                    | 15,400          | 3,889           | 21,211          | 12,626         |
| No. of MULs                         | 1                        | 1                          | 1                                        | 1               | 3               | 5               | 6              |
| Size of LUT                         | 128 B                    | 64 KB                      | 128 B                                    | 64 KB           | 4.3 KB          | 56KB            | 2.4 KB         |

The actual synthesis of the proposed architecture was performed using 28nm process with Synopsys design compiler for the accurate evaluation and the implementation result was compared with those of other designs [4], [5]. Table 2 shows the comparisons, in which the Case 1 of the proposed algorithm reduced the pipeline latency by 22% with an area increase of 34% compared to [4]. As for the Case 2 of the proposed algorithm compared to [5], its reduction of the pipeline latency about 11% with an area increase of 33%.

When the results shown in Table 2 are compared with the results in Table 1, the delay time reduction rate of Case 1 was increased by 6% from 16% to 22% compared to [4], and the hardware area was increased by 13% from 21% to 34%. The delay time reduction rate of Case 2 increased by 4% from 7% to 11% compared to [5], and the hardware area decreased by 12% from 45% to 33%. As a result of comparing the two tables, the hardware area sizes in Tables 1 and 2 were equally increased by an average of 33% compared to the previous algorithms [4] and [5].

In general, power consumption is correlated with area and is proportional to area. Therefore, area-timing products (ATP) using area and timing values are widely used in fields similar to this article and are used as indicators of power consumption through analysis of ATP results [17], [18]. Therefore, we compared and analyzed the previous algorithms through the ATP model in relation to power consumption.

Table 4 shows the comparison of the ATP results of the proposed algorithm with the previous algorithms [4] and [5]. In proposed cases 1 and 2, ATP was calculated by classifying LUT and multiplier in the first step. This is for accurate calculation because the delay time and power consumption of the two units are different. As a result of ATP analysis, it was confirmed that the power consumption of the proposed pipeline divider increased by about 37% which is similar to the increased hardware size of 33% compared to the previous algorithms. Through these results, it was confirmed that the area and power consumption are proportional.

Among previous works the pipelined division algorithm was used in [7]. There could be many differences between the proposed design and [7]. The main differences are how to calculate fractions and whether to perform error analysis on the accuracy of double-precision. In [7], since floating
B. COMPARISON WITH TRADITIONAL MULTIPLICATIVE ALGORITHMS

A multiplicative algorithm uses hardware integrated with a floating-point multiplier and LUT and calculates a quotient by approximation. Newton-Raphson and series-expansion algorithms are well known multiplicative algorithms. An important disadvantage of these algorithms is their long latency; they are also difficult to be pipelined due to their iterative execution. For example, the LUT for a 16-bit seed Newton-Raphson or series-expansion divider is 64 KB [15]. These 16-bit seed dividers execute a division operation with three iterations in double-precision.

The comparison with the pipelinable division algorithms and traditional multiplicative algorithms is well discussed in [4]. Based on the comparison result of [4], we compare the proposed algorithm with traditional multiplicative algorithms, as shown in Table 3. Similar to [4], the function units of Newton-Raphson, series-expansion, and accurate quotient approximation algorithms are shown in the tables of [15] and [16].

The proposed algorithm has a larger area cost than Newton-Raphson, series-expansion, and accurate quotient approximation algorithms, but has a shorter latency and can be pipelined. Therefore, the proposed algorithm can be used effectively in systems where the divider is often used.

VI. CONCLUSION

In this article, a novel low latency pipelined divider architecture for double-precision numbers was proposed. We were able to reduce the pipeline depth by one step compared to the previous schemes. The proposed algorithm has been applied to the two conventional divider architectures and could reduce computational latency without increasing the LUT size. The proposed divider is suitable in systems where double-precision division is frequently required.

REFERENCES

[1] S. F. Oberman and M. J. Flynn, “Design issues in division and other floating-point operations,” IEEE Trans. Comput., vol. 46, no. 2, pp. 154–161, Feb. 1997.
[2] W. J. Lee, W. C. Park, V. P. Srimi, and T. D. Han, “Simulation development environment for mobile 3D graphics architectures,” IET Comput. Digit. Technol., vol. 1, no. 5, pp. 501–507, Sep. 2007.
[3] L. Chen, Q. Yang, and J. Shi, “An improved division algorithm with a small lookup table and its implementation,” in Proc. 3rd Int. Conf. Anti-Counterfeiting, Secur., Identificat. Commun., Aug. 2009, pp. 21–24.
[4] J.-C. Jeong, W.-C. Park, W. Jeong, T.-D. Han, and M.-K. Lee, “A cost-effective pipelined divider with a small lookup table,” IEEE Trans. Com- put., vol. 53, no. 4, pp. 489–495, Apr. 2004.
[5] S. Singh, J. Biswas, and S. Nandy, “A cost effective pipelined divider for double precision floating point number,” in Proc. IEEE 17th Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP), 2006, pp. 132–137.
[6] Z. Yu, L. Chunqiang, H. Wei, S. Huan, and L. Jing, “Design and optimization of a divider in pipelined processors based on quotient-try," in Proc. 12th IEEE Conf. Ind. Electron. Appl. (ICIEA), Jun. 2017, pp. 1061–1064.
[7] C. R. S. Hanuman, J. Kamala, and A. R. Aruna, “Implementation of multi-precision floating point divider for high speed signal processing applications,” J. Supercomput., vol. 75, no. 9, pp. 6038–6054, Jun. 2019.
[8] S. Oberman, G. Favor, and F. Weber, “AMD 3DNow! Technology: Architectures and implementations,” IEEE Micro, vol. 19, no. 2, pp. 37–48, Apr. 1999.
[9] P. Hung, H. Fahmy, O. Mencer, and M. J. Flynn, “Fast division algorithm with a small lookup table," in Proc. Conf. Rec. 33rd Asilomar Conf. Signals, Syst. Comput., vol. 2, May 1999, pp. 1465–1468.
[10] C. L. Wey, S. Y. Lin, and M. T. Shiae, “A pipelined divider with a small lookup table," in Proc. WSEAS Conf. Instrum., Meas., Circuits Syst., Apr. 2007, pp. 181–185.
[11] C. Nan Lyu and D. W. Matula, “Redundant binary booth recoding," in Proc. 12th Symp. Comput. Arithmetic, Jul. 1995, pp. 50–57.
[12] IEEE Standard for Binary Floating–Point Arithmetic, Standard ANSI/IEEE standard 754-1985, 1985.
[13] J.-A. Pineiro and J. D. Bruguera, “High-speed double-precision computation of reciprocal, division, square root, and inverse square root," IEEE Trans. Com- put., vol. 51, no. 12, pp. 1377–1388, Dec. 2002.
[14] J. A. Pineiro and J. D. Bruguera, “High-speed double-precision computation of reciprocal, division, square root, and inverse square root," Tech. Rep. 2001. [Online]. Available: http://www.ac.usc.es.
[15] S. F. Obermann and M. J. Flynn, “Division algorithms and implementa- tions," IEEE Trans. Comput., vol. 46, no. 8, pp. 833–845, Aug. 1997.
[16] D. Wong and M. J. Flynn, “Fast division using accurate quotient approxima- tions to reduce the number of iterations," IEEE Trans. Comput., vol. 41, no. 8, pp. 981–985, Aug. 1992.
[17] H. F. Urgardag, F. de Dinechin, Y. S. Gener, S. Goren, and L.-S. Didier, “Hardware division by small integer constants," IEEE Trans. Comput., vol. 66, no. 12, pp. 2097–2110, Dec. 2017.
M. Istoan and F. de Dinechin, “Automating the pipeline of arithmetic data-paths,” in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 704–709.

JUWON YUN was born in South Korea, in 1986. He received the B.S. degree from the Department of Game and Multimedia Engineering, Korea Polytechnic University, Siheung, South Korea, in 2013, and the M.S. and Ph.D. degrees in computer engineering from Sejong University, Seoul, South Korea, in 2020. His current research interests include sound tracing, game engine, mobile GPU, and computer graphics.

JINYOUNG LEE was born in South Korea, in 1988. He received the B.S. degree from the Department of Information Control Engineering, Kwang-Woon University, Seoul, South Korea, in 2013. He is currently pursuing the Ph.D. degree in computer engineering with Sejong University, Seoul. His current research interests include computer graphics, computer architecture, and machine learning.

WOO-NAM CHUNG received the B.S., M.S., and Ph.D. degrees in computer science from Yonsei University, South Korea. He is currently a Senior Engineer with Sejong University. His current research interests include real-time ray tracing GPUs, and 3D graphics algorithms and applications.

CHEONG GHIL KIM (Member, IEEE) received the B.S. degree in computer science from the University of Redlands, CA, USA, in 1987, and the M.S. and Ph.D. degrees in computer science from Yonsei University, South Korea, in 2003 and 2006, respectively. He is currently a Professor with the Department of Computer Science, Namseoul University, South Korea. His research interests include multimedia embedded systems, mobile AR, and 3D contents.

WOO-CHAN PARK received the B.S., M.S., and Ph.D. degrees in computer science from Yonsei University, Seoul, South Korea, in 1993, 1995, and 2000, respectively. From 2001 to 2003, he was a Research Professor with Yonsei University. He is currently a Professor of computer engineering with Sejong University, Seoul. His current research interests include ray tracing processor architecture, 3-D rendering processor architecture, real-time rendering, advanced computer architecture, computer arithmetic, lossless image compression hardware, and application-specific integrated circuit design.

* * *