Dual-Purpose Hardware Algorithms and Architectures – Part 2: Integer Division

Jihee Seo[1,2] and Dae Hyun Kim[2]
[1]Synopsys, Inc., Hillsboro, OR, USA
[2]School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA, USA
jiheeseo@synopsys.com, daehyun.kim@wsu.edu

***Abstract*—Integer division is different from floating-point division in that (1) the execution time of an integer division is highly dependent on the leading 1 locations of operands, (2) x and $-x$ have different magnitude parts if x is a two's complement integer, and (3) rounding is not necessary. In this paper, we apply the interval-analysis-based division algorithm proposed in "Dual-Purpose Hardware Algorithms and Architecture – Part 1: Floating-Point Division" [1] to offline and online integer division. We implement four online integer dividers using the algorithm, compare them with other dividers, and present detailed simulation results with in-depth analysis of the dividers. We find that the online dividers outperform the offline dividers when the waiting time for online operands goes up and the number of quotient bits to obtain goes down.**

***Index Terms*—Divider; Integer Arithmetic; Online Division;**

I. Introduction

Division is one of the most commonly-used arithmetic operations, so researchers have proposed efficient algorithms for high-performance division [2]–[6]. Most of them aim for offline division, whereas some division algorithms have been proposed for online division [7]–[13]. In the paper "Dual-Purpose Hardware Algorithms and Architectures – Part 1: Floating-Point Division", a companion to this paper, we have proposed a dual-purpose (offline and online) algorithm for floating-point division. The algorithm works as an offline algorithm if both the dividend and the divisor are fully given at cycle 0. On the contrary, if one or both of the operands are partially given at cycle 0 and additionally given later, the algorithm works as an online algorithm and starts generating quotient digits before receiving all the digits of the operands.

Digit-recurrence integer and floating-point division algorithms are generally based on the same or similar principles. However, the execution time of an integer division varies depending on the leading 1 locations of operands. In addition, for a nonzero integer x, the magnitude parts of x and $-x$ are different in the two's complement representation. Moreover, integer division does not need rounding and normalization.

In this paper, we apply the dual-purpose hardware algorithm to integer division and propose hardware architectures for that. Our contributions in this paper are as follows:

- The division algorithm we propose uses the conventional non-redundant binary (normal binary) number system, so it does not require hardware for the conversion between redundant and non-redundant binary number systems.
- The algorithm and the hardware architectures can be used for both offline and online division.
- For online division, the division algorithm fully utilizes all the given dividend bits. In addition, if the divisor of a division is fully given at cycle 0, the division algorithm fully utilizes the divisor. These two features help reduce the execution time significantly.
- The online division algorithm can be used for signed integers with a slight modification.

The rest of this paper is organized as follows. In Section II, we review the digit-recurrence-based division algorithm for unsigned integers and present an equivalent form of the algorithm using interval analysis. In Section III, we explain interval-analysis-based normal-binary online division algorithms for unsigned and signed integers. Section IV shows hardware implementation of the algorithms. We compare several offline and online dividers in Section V and conclude in Section VI.

II. Normal-Binary Offline Division

In this section, we briefly review the non-restoring offline division algorithm for unsigned integer division. More details can be found in [1], [14].

A. *Offline Normal-Binary Unsigned Integer Division*

For $x = d \cdot q + rem$ where x, d, q, and rem are the dividend, the divisor, the quotient, and the remainder of the unsigned integer division, respectively, it should satisfy the following:

$$0 \le rem < d. \tag{1}$$

The quotient obtained until iteration j is as follows:

$$q[j] = q_{n-1}q_{n-2}\cdots q_{n-j} = \sum_{i=n-j}^{n-1} q_i \cdot r^i, \tag{2}$$

where $q_i \in \{0, 1, \cdots, r-1\}$.

For $x = d \cdot q + rem$, $q[j]$ should satisfy the following [14]:

$$0 \le x - d \cdot q[j] < d \cdot r^{n-j} \Leftrightarrow 0 \le r^{j-n} \cdot (x - d \cdot q[j]) < d. \tag{3}$$

Similarly, $q[j+1]$ should satisfy the folowing:

$$\begin{aligned} &0 \le r^{j+1-n} \cdot (x - d \cdot q[j+1]) < d \\ \Leftrightarrow\ &0 \le r^{j+1-n} \cdot (x - d \cdot q[j]) - d \cdot q_{n-j-1} < d. \end{aligned} \tag{4}$$

Define $w[j]$ as $w[j] = r^{j-n} \cdot (x - d \cdot q[j])$. Then, $q_{n-j-1} = k$ if the following is satisfied:

$$k \cdot d \le r \cdot w[j] < (k+1) \cdot d. \tag{5}$$

B. *Interval-Analysis-Based Offline Division*

For $x = d \cdot q + rem$, suppose we have obtained $q[j] = q_{n-1} \cdots q_{n-j}$. This means that x/d satisfies the following:

$$q[j] \le \frac{x}{d} < q[j] + r^{n-j}. \tag{6}$$

In this case, $q_{n-j-1} = k$ if the following condition is satisfied:

$$q[j] + k \cdot r^{n-j-1} \le \frac{x}{d} < q[j] + (k+1) \cdot r^{n-j-1}. \tag{7}$$

For example, suppose $n = 4$, $r = 2$, $x = 14$, $d = 2$, $j = 2$, so $q[2] = 0100_2 = 4_{10}$ in which q_1q_0 is set to 00. In this case, $q_1 = 0$ if $4 + 0 \cdot 2^1 \le x/d < 1 + 2 \cdot 2^1$ and $q_1 = 1$ if $4 + 1 \cdot 2^1 \le x/d < 4 + 2 \cdot 2^1$.

Rearranging the terms in (7) leads to (5), which means the quotient digit selection based on interval analysis in (7) is equivalent to the digit-recurrence division algorithm in (5).

C. Dependency on the Leading 1 Locations of the Operands (Speed-Up Technique)

Suppose x and d are given and let the locations (indices) of the leading 1 *bits* of x and d are l_x and l_d, respectively. If $l_x \le l_d$, then q is 0 or 1. If $l_x > l_d$, some of the most significant bits (MSBs) of q are 0 as follows:

$$2^{l_x - l_d - 1} = \frac{2^{l_x}}{2^{l_d+1}} < \frac{x}{d} < \frac{2^{l_x+1}}{2^{l_d}} = 2^{l_x - l_d + 1}. \tag{8}$$

Thus, $q_{n-1} \cdots q_{l_x - l_d + 1}$ is $0 \cdots 0$. In this case, we can start the division from $q_{l_x - l_d}$, which is a well-known technique.

III. Normal-Binary Online Division

In this section, we propose an interval-analysis-based normal binary online division algorithm. More details can be found in [1].

A. Normal-Binary Unsigned Integer Online Division

For $x = d \cdot q + rem$, suppose two unsigned integers x and d are partially given from the MSBs until iteration j and we have obtained q from them as follows:

$$x[j] = x_{n-1}x_{n-2} \cdots x_{a[j]}(0 \cdots 0) = \sum_{i=0}^{a[j]} x_i \cdot r^i, \tag{9}$$

$$d[j] = d_{n-1}d_{n-2} \cdots d_{b[j]}(0 \cdots 0) = \sum_{i=0}^{b[j]} d_i \cdot r^i, \tag{10}$$

$$q[j] = q_{n-1}q_{n-2} \cdots q_{c[j]}(0 \cdots 0) = \sum_{i=0}^{c[j]} q_i \cdot r^i, \tag{11}$$

where $a[j]$ and $b[j]$ are the indices of the rightmost digits of x and d given until iteration j, respectively, and $c[j]$ is the index of the rightmost digit of q obtained until iteration j. We also show 0's in the parentheses for the ungiven digits of x and d. The range of x/d is as follows:

$$\Big[\Big(\frac{x}{d}\Big)_{\text{MIN}} = \frac{x[j]}{d[j] + r^{b[j]} - 1}, \Big(\frac{x}{d}\Big)_{\text{MAX}} = \frac{x[j] + r^{a[j]} - 1}{d[j]}\Big]. \tag{12}$$

In this case, $q_{c[j]-1} = k$ if the following is satisfied as shown in Fig. 1(a):

$$q[j] + k \cdot r^{c[j]-1} \le \Big(\frac{x}{d}\Big)_{\text{MIN}}, \tag{13}$$

$$\Big(\frac{x}{d}\Big)_{\text{MAX}} < q[j] + (k+1) \cdot r^{c[j]-1}. \tag{14}$$

However, notice that $(x/d)_{\text{MIN}}$ and $(x/d)_{\text{MAX}}$ might not fall into the same sub-range as shown in Fig. 1(b). In this case, we need more digits of x and/or d to find $q_{c[j]-1}$.

We rewrite (13) as follows:

$$0 \le x[j] - (q[j] + k \cdot r^{c[j]-1}) \cdot (d[j] + r^{b[j]} - 1). \tag{15}$$

Similarly, we rewrite (14) as follows:

$$x[j] + r^{a[j]} - 1 - d[j] \cdot (q[j] + (k+1) \cdot r^{c[j]-1}) < 0. \tag{16}$$

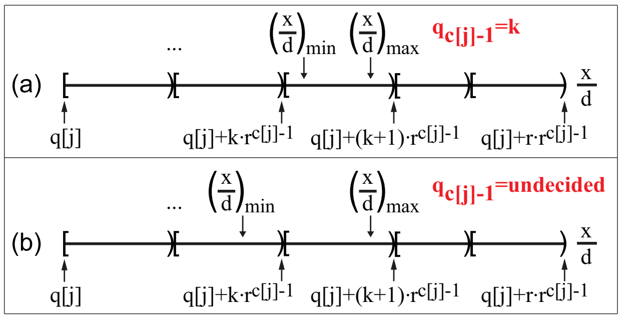
(a)
...
$\left(\frac{x}{d}\right)_{\min}$
$\left(\frac{x}{d}\right)_{\max}$
$q_{c[j]-1}=k$
$\frac{x}{d}$
q[j]
$q[j]+k \cdot r^{c[j]-1}$
$q[j]+(k+1) \cdot r^{c[j]-1}$
$q[j]+r \cdot r^{c[j]-1}$
(b)
...
$\left(\frac{x}{d}\right)_{\min}$
$\left(\frac{x}{d}\right)_{\max}$
$q_{c[j]-1}$=undecided
$\frac{x}{d}$
q[j]
$q[j]+k \cdot r^{c[j]-1}$
$q[j]+(k+1) \cdot r^{c[j]-1}$
$q[j]+r \cdot r^{c[j]-1}$

Fig. 1. Interval-analysis-based unsigned integer online division

Evaluation of (15) and (16) requires shifted versions of $q[j]$ and $d[j]$, and $d[j] \cdot q[j]$. Since the multiplication of $d[j]$ and $q[j]$ is costly and cannot be completed in a cycle, we incrementally update $d[j] \cdot q[j]$ as follows. If $m[i,j] = (\sum_{p=n-i}^{n-1} d_p \cdot r^p) \cdot (\sum_{p=n-j}^{n-1} q_p \cdot r^p)$, then the following holds:

$$\begin{aligned} m[i+w, j+t] = m_{[}i,j] &+ (\sum_{p=n-i-w}^{n-i-1} d_p \cdot r^p) \cdot (\sum_{p=n-j}^{n-1} q_p \cdot r^p) \\ &+ (\sum_{p=n-i-w}^{n-1} d_p \cdot r^p) \cdot (\sum_{p=n-j-t}^{n-j-1} q_p \cdot r^p). \end{aligned} \tag{17}$$

The details of when the incremental update is executed is shown in Section IV.

B. Example: Radix-4 Online Division

Suppose $n = 4$, $r = 4$, $x = 3320_4$, and $d = 0031_4$. We also assume that one digits of x and d are given every clock cycle starting from the MSBs.

At cycle 1, $x_3 = 3$ and $d_3 = 0$ are given. Applying (15) and (16) for $k = 0, 1, 2, 3$ leads to (true, false) for $k = 0$ and (false, false) for the others, so we cannot find q_3. At cycle 2, $x_2 = 3$ and $d_2 = 0$ are given. None of $k = 0, 1, 2, 3$ satisfies (15) and (16) at the same time.

At cycle 3, $x_1 = 2$ and $d_1 = 3$ are given. Applying (15) and (16) for $k = 0, 1, 2, 3$ leads to (true, true) for $k = 0$ and (false, true) for the others. As a result, $q_3 = 0$. At cycle 4, $x_0 = 0$ and $d_0 = 1$ are given. Only $k = 0$ satisfies both (15) and (16), so $q_2 = 1$.

Since all the digits of x and d have been given, we can certainly obtain q_1 in the next cycle. At cycle 5, applying the inequalities leads to (true, true) only for $k = 0$, so $q_1 = 0$. At cycle 6, the evaluation of the inequalities results in (true, false) for $k = 0, 1, 2$ and (true, true) for $k = 3$, so $q_0 = 3$. $x/d = 3320_4/0031_4 = 248_{10}/13_{10} = 19_{10} = 0103_4$, so the answer is correct.

C. Online Division with Offline Operands

Suppose x and d are fully given at cycle 0. In this case, $a[j] = 0$ and $b[j] = 0$ for all j and $c[j] = n - j$ in (9)–(11). Then, we can rewrite (15) as follows:

$$k \cdot d \le r \cdot \{r^{-c[j]} \cdot (x - d \cdot q[j])\}, \tag{18}$$

which is equivalent to the left side of (5). Similarly, we can rewrite (16) as follows:

$$r \cdot \{r^{-c[j]} \cdot (x - d \cdot q[j])\} < (k+1) \cdot d, \tag{19}$$

TABLE I
AN EXAMPLE OF THE ONLINE DIVISION ALGORITHM. $n = 4$, $r = 4$, $x = 3320_4$, AND $d = 0031_4$. WE ASSUME THAT ONE DIGITS OF x AND d ARE GIVEN EVERY CLOCK CYCLE.

Cycle	x	d	q
0	XXXX$_4$	XXXX$_4$	XXXX$_4$
1	3XXX$_4$	0XXX$_4$	XXXX$_4$
	Apply (15) and (16). $k = 0$ (true, false), $k = 1$ (false, false), $k = 2$ (false, false), $k = 3$ (false, false),		
2	33XX$_4$	00XX$_4$	XXXX$_4$
	Apply (15) and (16). $k = 0$ (true, false), $k = 1$ (false, false), $k = 2$ (false, false), $k = 3$ (false, false),		
3	332X$_4$	003X$_4$	XXXX$_4$
	Apply (15) and (16). $k = 0$ (true, true), $k = 1$ (false, true), $k = 2$ (false, true), $k = 3$ (false, true) $\Rightarrow q = 0XXX_4$		
4	3320$_4$	0031$_4$	0XXX$_4$
	Apply (15) and (16). $k = 0$ (true, false), $k = 1$ (true, true), $k = 2$ (false, true), $k = 3$ (false, true) $\Rightarrow q = 01XX_4$		
5	3320$_4$	0031$_4$	01XX$_4$
	Apply (15) and (16). $k = 0$ (true, true), $k = 1$ (false, true), $k = 2$ (false, true), $k = 3$ (false, true) $\Rightarrow q = 010X_4$		
6	3320$_4$	0031$_4$	010X$_4$
	Apply (15) and (16). $k = 0$ (true, false), $k = 1$ (true, false), $k = 2$ (true, false), $k = 3$ (true, true) $\Rightarrow q = 0103_4$		

which is equivalent to the right side of (5). This proves that the online divider will work as an offline divider (*dual-purpose*) if the operands are offline (fully given at cycle 0).

D. Normal-Binary Signed Integer Online Division

In this section, we briefly show how to use the proposed division algorithm for signed integer online division. For $x = d \cdot q + rem$, we assume x and rem have the same sign. Then we find the following formulas for q and rem:

$$q = (-1)^{s_q} \cdot \left\lfloor \frac{|x|}{|d|} \right\rfloor, \tag{20}$$

$$rem = (-1)^{s_{rem}} \cdot (|x| - |q| \cdot |d|). \tag{21}$$

where $s_q = 0$ if $x \cdot d > 0$ and $s_q = 1$ if $x \cdot d < 0$, and $s_{rem} = 0$ if $x > 0$ and $s_{rem} = 1$ if $x < 0$. Thus, signed integer division can be completed in three steps as follows:

- Computation of $|x|$ and $|d|$.
- Unsigned integer division: $q' = \lfloor |x|/|d| \rfloor$.
- Sign conversion: $q = q'$ if $s_q = 0$. $q = \overline{q'}$ if $s_q = 1$.

For online division, we need the following modifications. If $x[j] = x_{n-1} \cdots x_{a[j]}$ and $d[j] = d_{n-1} \cdots d_{b[j]}$ are given and they are negative (i.e., $x_{n-1} = 1$, $d_{n-1} = 1$), we obtain the following ranges for $|x|$ and $|d|$:

$$|x|_{\text{MIN}} : \overline{x[j]} + 1, \quad |x|_{\text{MAX}} : \overline{x[j]} + r^{a[j]},$$
$$|d|_{\text{MIN}} : \overline{d[j]} + 1, \quad |d|_{\text{MAX}} : \overline{d[j]} + r^{b[j]},$$

where $\overline{x[j]}$ and $\overline{d[j]}$ are $\overline{x_{n-1} \cdots x_{a[j]}}0 \cdots 0$ and $\overline{d_{n-1} \cdots d_{b[j]}}0 \cdots 0$, respectively. In this case, we use the following inequalities for (15):

$$x > 0, d > 0 : \quad (q[j] + k \cdot r^{c[j]-1}) \times (d[j] + r^{b[j]-1}) \leq x[j]$$
$$x > 0, d < 0 : \quad (q[j] + k \cdot r^{c[j]-1}) \times (\overline{d[j]} + r^{b[j]}) \leq x[j]$$
$$x < 0, d > 0 : \quad (q[j] + k \cdot r^{c[j]-1}) \times (d[j] + r^{b[j]} - 1) \leq \overline{x[j]} + 1$$
$$x < 0, d < 0 : \quad (q[j] + k \cdot r^{c[j]-1}) \times (\overline{d[j]} + r^{b[j]}) \leq \overline{x[j]} + 1$$

We also use the following inequalities for (16):

$$x > 0, d > 0 : \quad x[j] + r^{a[j]} - 1 < (q[j] + (k+1) \cdot r^{c[j]-1}) \times d[j]$$
$$x > 0, d < 0 : \quad x[j] + r^{a[j]} - 1 < (q[j] + (k+1) \cdot r^{c[j]-1}) \times (\overline{d[j]} + 1)$$
$$x < 0, d > 0 : \quad (\overline{x[j]} + r^{a[j]}) < (q[j] + (k+1) \cdot r^{c[j]-1}) \times d[j]$$
$$x < 0, d < 0 : \quad (\overline{x[j]} + r^{a[j]}) < (q[j] + (k+1) \cdot r^{c[j]-1}) \times (\overline{d[j]} + 1)$$

IV. HARDWARE IMPLEMENTATION

In this section, we present the hardware implementation of the n-bit radix-r online divider.

A. Design Parameters

Table II shows two design parameters for the hardware implementation of the divider. r is the radix of the divider and $\log_2 r$ is the number of quotient bits to obtain every cycle. In this paper, we use 4 and 16 for r. For w, suppose $d[j] = d_{n-1} \cdots d_{b[j]}$ and we have $m[b[j], c[j]]$. In the next cycle, suppose $d_{b[j]-1} \cdots d_{b[j]-s}$ is additionally given ($s \geq w$). Then, we use only w digits of them ($d_{b[j]-1} \cdots d_{b[j]-w}$) to update m. Increasing w will reduce the clock frequency because the update of m in (17) will take more time. However, it will increase the probability of finding quotient digits in a cycle. In this paper, we use 2 and 4 for w.

B. Hardware Implementation: Overall Architecture

Fig. 2 shows the top-level architecture of the proposed online divider. The input consists of dividend x, divisor d, and their valid bits v_x and v_d sent from the senders of x and d, respectively. For example, if only $x_{n-1} \cdots x_{a[j]}$ is valid, v_x is $1 \cdots 10 \cdots 0$ in which there are $n - a[j]$ 1's and $a[j]$ 0's.

Path 1 detects leading 1's in x and d. If x and/or d does not have a leading 1, $F1$ is set to false and the state of the divider stays in Path 1. If $d = 0$, ERR is set to true and the division ends. If both x and d have leading 1's, we apply the speed-up technique and find $v_{q,1}$, $c_{q,1}$, and $c_{d,1}$, which are passed to v_q, c_q, and c_d through the mux, and move on to Path 2 in the next cycle. v_q is an n-bit valid signal for q (similar to v_x and v_d). c_d and c_q are the indices for $m[c_d, c_q]$. In Path 2, we evaluate (15) and (16), find quotient digits, and update m, v_q, c_q, and c_d, the last three of which are passed to the mux for the use in Path 2 again in the next cycle. Table III shows the internal registers. A state controller controls the mux in Fig. 2.

C. Hardware Implementation: Path 1

Fig. 3 shows Path 1 of the proposed divider. F_x (or F_d) is 1 if x (or d) has a leading 1 and l_x (or l_d) has the group index of the leading 1, which is the bit index of the leading 1 of x (or d) divided by $\log_2 r$. $F1$ is $F_x \cdot F_d$. If all the bits of d are valid and $d = 0$, ERR is set to 1 indicating a divide-by-zero error. From now on, we assume that $F1 = 1$ and $ERR = 0$.

D_L is $l_x - l_d$. If $D_L < 0$, then $x < d$, so we set v_q to $1 \cdots 1$ and the division ends ($q = 0$). If $D_L \geq 0$, we apply the speed-up technique to fill up q with leading 0's. $c_{q,1}$ is set to $(D_L + 1)$ because we have obtained $q_{n-1} \cdots q_{n-(\log_2 r) \cdot (D_L+1)}$ (all 0's) by the speed-up technique.

For $c_{d,1}$, we find the index t of the rightmost valid bit of d using v_d and set $c_{d,1}$ to $\lceil t/w \rceil$. Notice that we do not compute m in Path 1 because the quotient bits we find in Path 1 are all zero, so m is zero. Thus, we just find $c_{q,1}$ and $c_{d,1}$.

D. Hardware Implementation: Path 2

Path 2 consists of three steps as shown in Fig. 4.

TABLE II
DESIGN PARAMETERS AND CONSTANTS. n: DATA WIDTH.

	Description
r	Radix-r division (4 or 16)
w	# unused bits of d used to update m (2 or 4 bits) in (17)
n_r	$\lceil \log_2(n/\log_2 r) \rceil + 1$
n_w	$\lceil \log_2(n/w) \rceil$

TABLE III
INTERNAL REGISTERS

Reg	# bits	Description
q	n	Stores the quotient.
v_q	n	Stores the valid bits of q.
m	n	Stores $m[p,s]$.
c_d	n_w	Stores the first index of $m[c_d, c_q]$.
c_q	n_r	Stores the second index of $m[c_d, c_q]$.

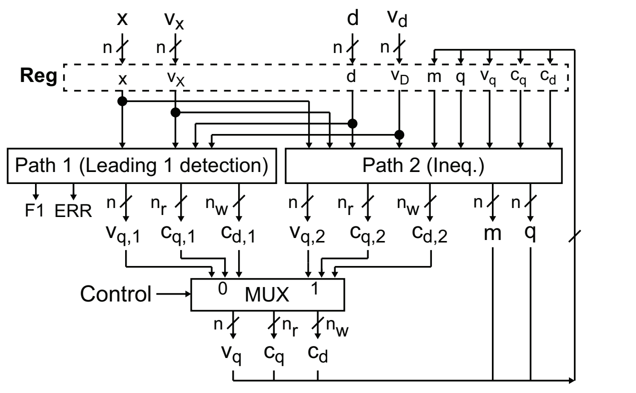

Fig. 2. The top-level architecture of the proposed divider for n-bit radix-r high-throughput division. $n_r = \lceil \log_2(n/\log_2 r) \rceil + 1$, $n_w = \lceil \log_2(n/w) \rceil$.

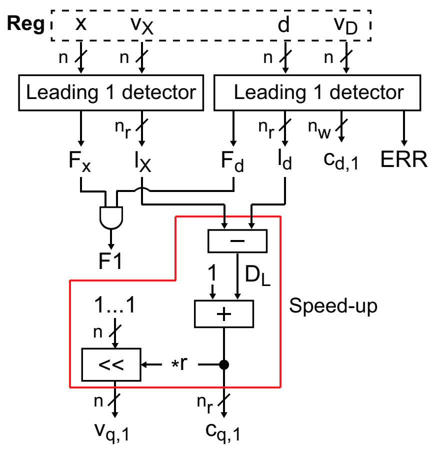

Fig. 3. Path 1: Leading 1 detection and the speed-up technique.

1) Step 1: Update m using (17).: We first find the index t of the rightmost valid bit of d using v_d and set g_d to $\lceil t/w \rceil$. If $c_d - g_d \geq 1$, d has at least w bits not included in m, so we update $m[w \cdot c_d, (\log_2 r) \cdot c_q]$ using (17) and decrease c_d by 1.

2) Step 2: Solve (15) and (16).: We solve (15) and (16) for $k = 0, \cdots, r-1$. Notice that most of the terms in (15) and (16) are shifted versions of $q[j]$, $d[j]$, and k, thus we use shifters for them. For the evaluation of the inequalities, we add the terms by carry-save adders (CSAs) and then use a carry-propagate adder (CPA) to obtain the final sum. We also pre-compute m_k for each k using (17) to reduce the computation time.

3) Step 3: Update m using (17).: If a certain k satisfies both (15) and (16), then we use a mux to select the pre-computed m_k for m. We also update q, $v_{q,2}$, and $c_{q,2}$ accordingly (shown as "Proc. q, v_q, c_q" in Fig. 4). If none of the inequality sets is true, then we do not update m, $c_{q,2}$, and $v_{q,2}$.

V. SIMULATION RESULTS

In this section, we present simulation results for 64-bit unsigned integer division. We implemented all the dividers using Verilog and synthesized them using Synopsys Design Compiler.

A. Dividers Used for the Comparison

We compare seven offline and five online dividers for the execution of an offline division. We include four floating-point dividers because they can also be used for integer division with slight modifications.

AN16 is a radix-8 floating-point divider [2]. It uses the radix-8 digit-recurrence algorithm with a look-up table for quotient digit selection. SA17 is a radix-16 floating-point divider [3]. It is based on a radix-16 digit-recurrence algorithm with a wide digit set. JB20 is a radix-64 floating-point divider [4]. It performs three radix-4 iterations to obtain six quotient bits in a cycle. NT05-4 and NT05-16 are the radix-4 and radix-16 integer dividers, respectively, proposed in [15]. Q2 and Q4 are the radix-4 and radix-16 offline dividers, respectively, based on (7).

AT03 is a radix-4 floating-point online divider [13]. It unfolds radix-2 recurrence equations to obtain two quotient bits in a cycle. QDxw are our online dividers and we call them QD dividers below. x is $\log_2 r$, the number of quotient bits to obtain in a cycle, and w is the number of divisor bits used to update m using (17).

B. Offline Division

First, we compare the dividers for an offline division. Table IV shows the clock periods, execution times, energy consumption, and areas of the dividers. Notice that we did not use the speed-up technique in any of the dividers for a fair comparison.

1) Execution Time: Table IV shows the execution times of the dividers. Q4 has the shortest execution time for an offline division. NT05-4, NT05-16, AN16, SA17, and JB20 are 98%, 88%, 65%, 116%, and 82% slower than Q4, respectively. Comparing Q4 and JB20, Q4 obtains four quotient bits per cycle while JB20 obtains six quotient bits per cycle. However, the clock period of Q4 is 49% shorter than that of JB20, so Q4 outperforms JB20 by 45%. We find similar trends in the other designs. Q2 also outperforms all the designs except Q4. In addition, even QD42 and QD44 outperform most of the other

TABLE IV
A COMPARISON OF THE DIVIDERS FOR A 64-BIT OFFLINE DIVISION. THE NUMBERS IN THE PARENTHESES ARE THE RATIOS OF THE RESULTS TO THE VALUES OF THE Q2 DESIGN.

	Offline dividers							Online dividers				
	AN16 [2]	SA17 [3]	JB20 [4]	NT05-4 [15]	NT05-16 [15]	Q2	Q4	AT03 [13]	QD22	QD24	QD42	QD44
Radix	8	16	64	4	16	4	16	4	4	4	16	16
Clock period (ns)	0.71	1.21	1.36	0.54	1.00	0.52	0.70	0.63	0.93	0.95	1.11	1.21
Execution time (ns)	18.46 (1.11)	24.20 (1.45)	20.40 (1.23)	22.14 (1.33)	21.00 (1.26)	16.64 (1.00)	11.20 (**0.67**)	25.83 (1.55)	29.76 (1.79)	30.40 (1.83)	17.76 (1.07)	19.36 (1.16)
Energy (pJ)	58.7 (2.47)	125.1 (5.27)	72.6 (3.06)	41.4 (1.74)	28.4 (1.19)	23.7 (**1.00**)	95.6 (4.03)	142.8 (6.02)	202.9 (8.55)	300.5 (12.66)	490.6 (20.67)	499.0 (21.02)
Area (um^2)	3,871 (1.34)	12,663 (4.37)	6,489 (2.24)	1,837 (**0.63**)	2,846 (0.98)	2,895 (1.00)	13,391 (4.63)	4,696 (1.62)	9,818 (3.39)	11,810 (4.08)	35,153 (12.14)	38,395 (13.26)

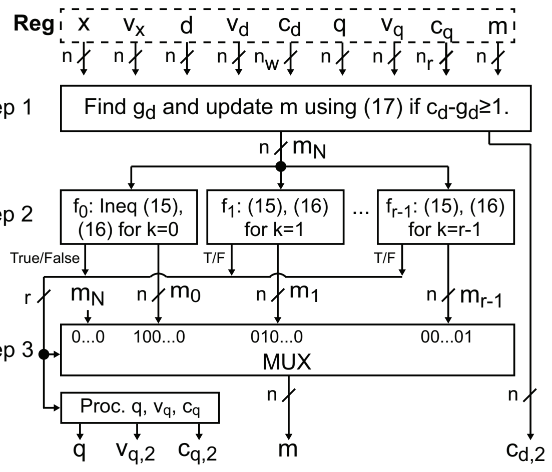

Fig. 4. Path 2: Evaluation of (15) and (16).

designs. On the other hand, QD22 and QD24 are slower than the others because they need 32 cycles to finish a division.

The reason that the clock period of Q4 is only 35% longer than that of Q2 is as follows. The most time-consuming part in the Q2 and Q4 dividers is the calculation of (7), which is composed of several CSA and a CPA stages. However, obtaining two more quotient bits per cycle in Q4 compared to Q2 requires adding two more 64-bit operands in (7), which increases the critical path delay only by one CSA delay. Q4 also uses 16:1 muxes (compared to 4:1 muxes in Q2), so its critical path delay is $0.18ns$ longer than that of Q2. However, Q4 spends only 16 cycles, whereas Q2 spends 32 cycles for the division. As a result, the execution time of Q4 is 33% shorter than that of Q2. The four QD designs have longer clock periods than Q4 because they have more complex logic and CSA stages for the computation of (15) and (16).

The following shows the critical path delays of Q4 and QD44.

$$\begin{aligned} \text{Q4}: &\quad 3 \text{ CSAs } (290ps) + \text{CPA } (200ps) + 4:1 \text{ MUX } (210ps) \\ \text{QD44}: &\quad \text{Ext\_D } (230ps) + 5 \text{ CSAs } (440ps) + \text{CPA } (180ps) \\ &\quad +16:1 \text{ MUX } (360ps) \end{aligned}$$

where "Ext_D" extracts four new bits in the divisor.

2) Energy Consumption and Area: Although Q4 shows the shortest execution time, it consumes more energy than some of the other designs. For example, Q4 consumes 63% more energy than AN16. On the other hand, Q2 consumes the least amount of energy among them. The reason that Q4 consumes much more energy than Q2 is that Q4 evaluates 15 inequalities whereas Q2 evaluates only three inequalities. For the same reason, the area of Q4 is $4.63\times$ as large as that of Q2. The four QD dividers consume even more energy and silicon area than Q4 because the QD dividers need additional hardware resources to update m, add more operands in the CSA stages for the inequality evaluation, and compute more inequalities.

Fig. 5 shows normalized area vs. normalized execution time of the dividers for an offline division. Both the area and execution time values are normalized to the Q2 design.

C. Online Division

Now we compare the integer dividers for online division. We do not include the NT05 designs because Q2 and Q4 have shorter execution time than NT05. Q2 and Q4 cannot process incomplete operands, so if an online division is given, they should wait until the operands are fully given.

1) Simulation Methodology: A *target divider* is the divider computing an online division. An online operand is generated in a *generator* and sent to the target divider starting from the MSB. Any unit could be used for the generator, but we use Q2 and Q4 for that in this paper. Thus, an online operand is the result of an offline division computed in a Q2 or Q4 divider. Since a division has two operands, we define *dependency types* for online division as follows (a, b, c, d are offline operands.):

- Type-X: The dividend is online, but its divisor is offline. We denote it by $(a/b)/c$ where $e = a/b$ is computed in a generator and e/c is computed in a target divider.
- Type-D: The dividend is offline, but the divisor is online. We denote it by $a/(b/c)$ where $e = b/c$ is computed in a generator and a/e is computed in a target divider.
- Type-XD: Both the dividend and divisor are online. We denote it by $(a/b)/(c/d)$ where $e = a/b$ and $f = c/d$ are computed in two generators and e/f is computed in a target divider.

We also define the size (*large, medium, small*) of a 64-bit unsigned number x as follows:

- Large: At least one of the eight MSBs is 1 ($x \geq 2^{56}$).
- Medium: $2^{24} \leq x < 2^{32}$.
- Small: $x < 2^{8}$.

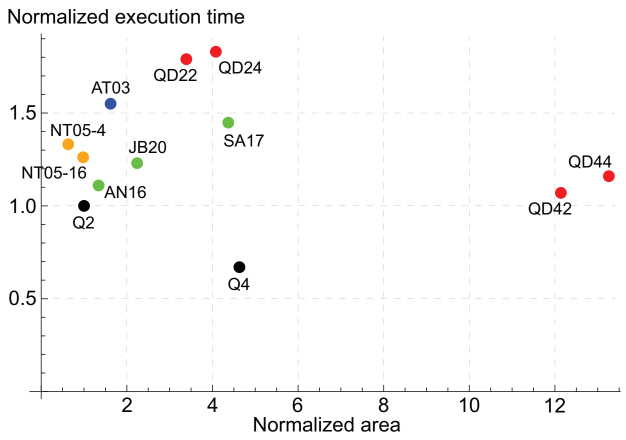

Fig. 5. Area vs. execution time for an offline division (Table IV). Both the areas and execution times are normalized to the Q2 design.

A simulation set for Type-X consists of two divisions, $e = a/b$ and $q = e/c$. a, b, and c are randomly generated. However, there are some trivial cases such as dividing a small dividend by a large divisor (so the quotient is zero). To avoid trivial cases, a is set to a large number (L), and b and c are set to small numbers (S), which is denoted by $(L/S)/S$. Similarly, we simulate $L/(L/S)$ and $L/(L/M)$ for Type-D. For Type-XD, we simulate $(L/S)/(L/M)$ and $(L/S)/(L/L)$.

For each dependency type, we generated and ran 10,000 simulation sets, obtained the cycle count of each set, and computed the sum of the execution times. Each offline division is computed in a Q2 or Q4 divider (data generator). The quotient of a generator is passed to a target divider computing an online division. We start and stop counting the number of cycles for each online division when the target divider receives the first bit of the online operand and generates the last bit of the quotient of the online division, respectively.

The execution of an online division is decomposed into four computation stages as follows:

- *Wait*: When an online divider computes an online division, it waits until a sufficient number of operand bits are given (*online delay*). An offline divider waits until the operands are fully given. This stage is the *Wait* stage.
- *Speed up*: When an integer divider receives leading 1's in both operands, it quickly fills up some of the quotient bits with 0's in Path 1 and switch to Path 2 from the next cycle. This stage is the *Speed-up* stage and always takes one cycle.
- *Ineq. (not found)* and *Ineq. (found)*: When the online dividers compute (15) and (16), they might or might not be able to find quotient bits. The former and the latter are denoted by *Ineq. (found)* and *Ineq. (not found)*, respectively.

2) Type-X Division: Fig. 6(a) shows a flowchart of the Type-X division and Fig. 6(b) shows the execution times of the dividers normalized to Q2 when the generator is a Q2 divider (i.e., the target divider receives two dividend bits every cycle except the first cycle). The offline dividers spend a large

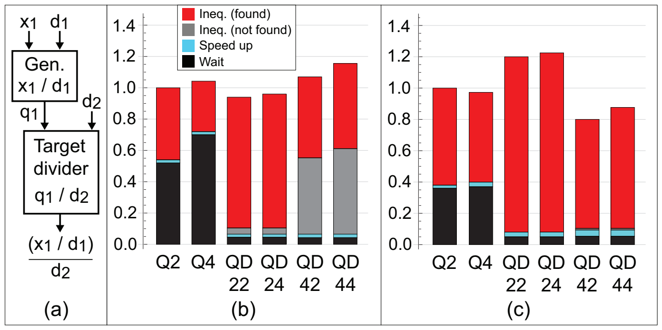

Fig. 6. (a) A flowchart of a Type-X division. The execution times of the dividers for Type-X $(L/S)/S$ division (normalized to Q2). (b) The generator is a Q2 divider, (c) The generator is a Q4 divider.

amount of time in the Wait stage because they start the division only when the operands are fully given.

Q2 is 4% faster than Q4, and QD22 and QD24 are 6% and 4% faster than Q2, respectively. QD22 and QD24 spend only one or two cycles in the Wait stage and move on to the inequality stage, whereas the offline dividers spend almost 29 cycles in the Wait stage. Notice that the divisors of the online divisions are offline, so the QD designs can fully utilize the divisors from cycle 0. Thus, QD22 and QD24 spend the same number of cycles for the divisions, but the clock period of QD22 is shorter than QD24. As a result, QD22 is 2% faster than QD24. For the same reason, QD42 is 8% faster than QD44. Moreover, QD22 and QD24 obtain the quotient bits most of the time because they try to find only two quotient bits at a time. On the contrary, QD42 and QD44 obtain the quotient bits only for about a half of the total cycles because receiving two bits of d per cycle is not enough to obtain four bits of q every cycle.

Fig. 6(c) shows the execution times when the generator is a Q4 divider. Since four bits of the dividends are given to the target divider every cycle, the offline dividers spend less time in the Wait stage. However, QD22 and QD24 do not benefit from receiving four bits of x every cycle because receiving two bits of x every cycle was enough to find two quotient bits every cycle. On the contrary, QD42 and QD44 outperform Q2 and Q4 by 6% to 31% because the former can obtain four quotient bits almost every cycle. In sum, QD22 and QD24 outperform the others when the generator sends two bits to them every cycle. However, if the generator sends four bits to them every cycle, QD42 and QD44 benefits from that significantly and outperform the others.

3) Type-D Division: Fig. 7(a) shows a flowchart of the Type-D division and Fig. 7(b) shows the execution times of the dividers for $L/(L/M)$ when the generator is a Q2 divider. The offline dividers still spend a long time in the Wait stage. On the contrary, the online dividers wait only for one or two cycles in the Wait stage. Similar to the Type-X division, QD22 and QD24 find quotient bits almost every cycle. However, QD42 and QD44 cannot obtain quotient bits for almost 50% of the clock cycles in the Inequality stage. This is again because receiving two divisor bits per cycle from the generator is not

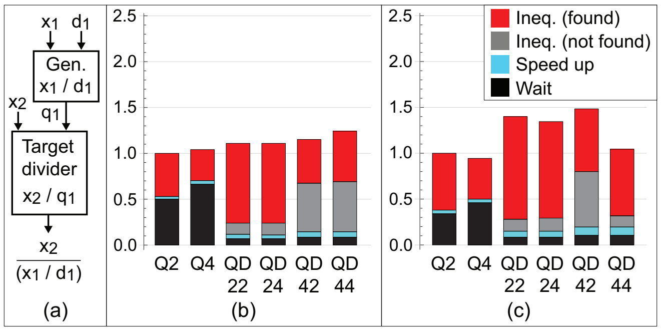

Fig. 7. (a) A flowchart of the Type-D division. The execution times of the dividers for Type-D $L/(L/M)$ division (normalized to Q2). (b) Generator: Q2, (c) Generator: Q4.

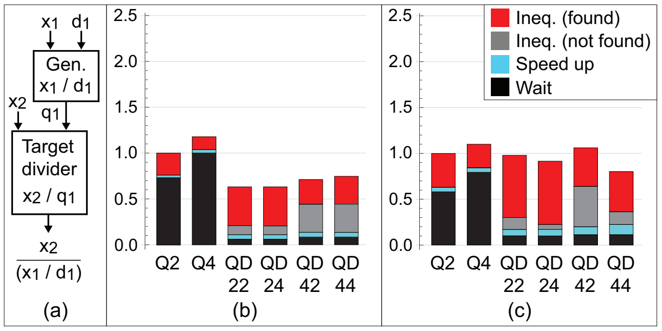

Fig. 8. (a) A flowchart of the Type-D division. The execution times of the dividers for Type-D $L/(L/S)$ division (normalized to Q2). (b) Generator: Q2, (c) Generator: Q4.

enough to obtain four quotient bits every cycle.

On the other hand, the Q2 and Q4 dividers outperform the others because the range of the quotient of $L/(L/M)$ is $[2^{24}, 2^{32}]$, so Q2 spends 12 to 16 cycles and Q4 spends 6 to 8 cycles in the Inequality stage. Since they have much shorter clock periods than the online dividers, the execution times of the offline dividers are shorter than those of the online dividers.

Fig. 7(c) shows the execution times of the dividers for $L/(L/M)$ when the generator is a Q4 divider. As shown in the figure, the execution time of QD44 goes down significantly because QD44 can find four quotient bits almost every cycle. However, the execution times of Q2 and Q4 also go down because the generator sends four bits to them every cycle, so the Q2 and Q4 dividers spend less time in the Wait stage.

Fig. 8(b) and (c) show the simulation results for Type-D $L/(L/S)$ division. The difference between $L/(L/S)$ and $L/(L/M)$ is that the generator computing L/S in the former spends more time than the one computing L/M in the latter. Thus, the offline dividers should wait longer in the $L/(L/S)$ case, which is why the execution times in the Wait stage in Fig. 8(b) and (c) occupy a significant portion of the total execution times compared to Fig. 7. However, the range of the quotient of $L/(L/S)$ is $[1, 2^16]$, so Q2 and Q4 spend less time in the Inequality stage. On the contrary, the QD dividers can still find some quotient bits while receiving the divisor bits, so they outperform the Q2 divider by 24% to 35% in Fig. 8(b). In Fig. 8(c), the generator finishes (L/S) in almost 16 cycles, so the offline dividers wait only for 16 cycles. As a result, the execution times of Q2 and Q4 go down and are comparable to those of the QD dividers.

4) Type-XD Division: There are two generators for the operands in the Type-XD division. The generators start their execution at cycle 0 and the target divider starts its execution when it receives the first bit of its dividend or divisor.

Fig. 9(b) and (c) show the execution times for $(L/S)/(L/M)$ when the generators for (dividend, divisor) are (Q2, Q2) and (Q2, Q4), respectively. In both cases, the QD dividers outperform the Q2 divider by 13% to 30%. We find that the number of maximally obtainable quotient bits in a cycle in this simulation set is approximately two. Thus, QD22 and QD24 outperform QD42 and QD44 by 8% to 15%.

Fig. 9(d) and (e) show the execution times for $(L/S)/(L/M)$ when the generators for (dividend, divisor) are (Q4, Q2) and (Q4, Q4), respectively. In Fig. 9(d), the divisions computed in the target dividers are L/M, so the target dividers need to compute about 36 quotient bits. Due to this, the performance of Q2 and Q4 is comparable to that of the QD dividers. In fact, the simulation in Fig. 9(d) is similar to that in Fig. 7(b) because the divisor in the target divider receives two bits. Although the dividend in the target divider receives four bits in Fig. 9(d), the divisor is the bottleneck in this case, so there is no big difference between Fig. 9(d) and Fig. 7(b). On the other hand, Fig. 9(e) shows a pattern slightly different from Fig. 7(c) because the dividend of a target division also comes from a generator in Fig. 9(e). In this case, QD44 shows the shortest execution time.

Fig. 10(b), (c), (d), and (e) show the execution times for $(L/S)/(L/L)$ when the data generators for (dividend, divisor) are (Q2, Q2), (Q2, Q4), (Q4, Q2), and (Q4, Q4), respectively. The target division is $(L/S)/(L/L) = L/S$ and the divisor L/L is computed in a few cycles, so this case is similar to the Type-X division. Thus, Fig. 10(b) and (c) are similar to Fig. 6(b). Similarly, Fig. 10(d) and (e) are similar to Fig. 6(c).

5) Summary: For online division, execution times of the dividers are highly dependent on the magnitudes of the operands, and how fast (bit rate) dependent operands receive their bits from the senders. We summarize our observation of the execution time trends as follows.

- Magnitude: Let l_x and l_d be the indices of the leading 1's of x and d, respectively. If l_x and l_d increase, the online dividers would outperform the offline dividers. The earlier the leading 1's appear in x and d, the earlier the online dividers can start the division.
- Bit rate: Let N_d be the average number of bits that the online operands of a target divider receive from the senders every cycle. If N_d increases, the offline dividers would outperform the online dividers.

The execution time of an online division is also directly related to the clock period of the target divider. Thus, it is very crucial to minimize the clock periods of the online dividers.

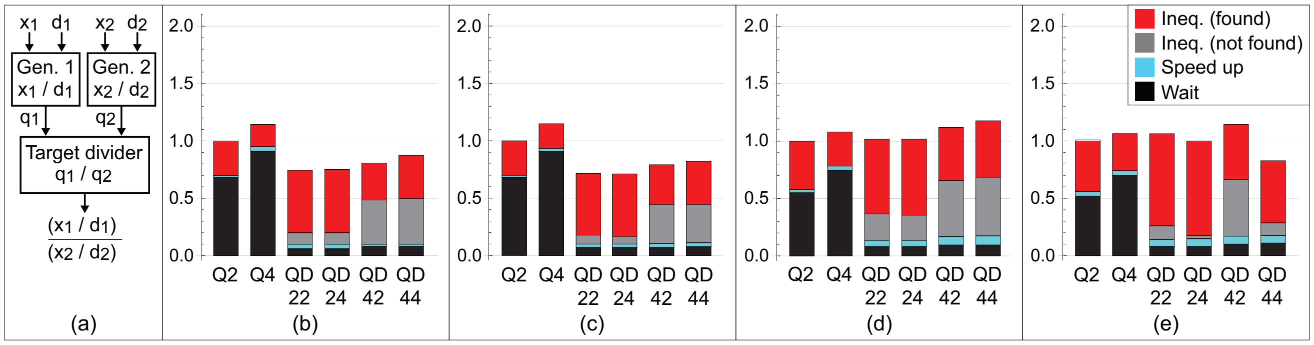

Fig. 9. (a) A flowchart of the Type-XD division. The execution times of the dividers for Type-XD $(L/S)/(L/M)$ division (normalized to Q2). (Generator for (L/S), Generator for (L/M)): (b) (Q2, Q2), (c) (Q2, Q4), (d) (Q4, Q2), (e) (Q4, Q4).

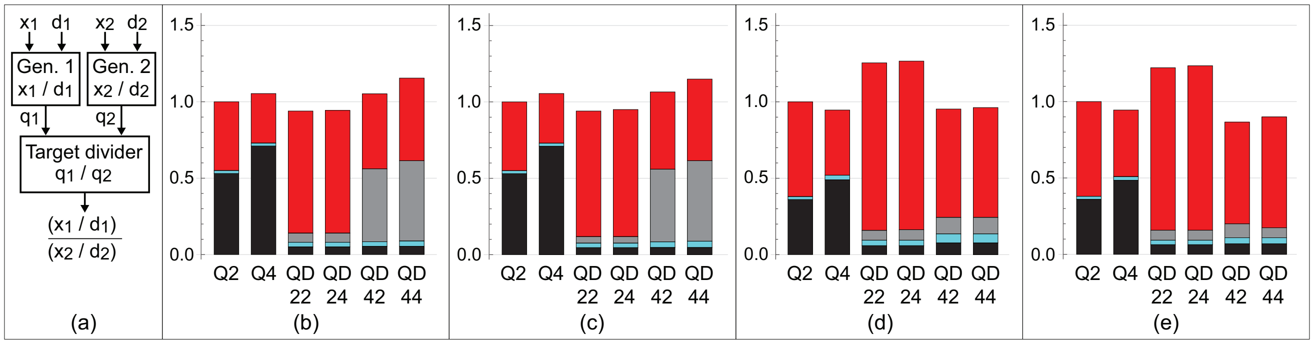

Fig. 10. (a) A flowchart of the Type-XD division. The execution times of the dividers for Type-XD $(L/S)/(L/L)$ division (normalized to Q2). (Generator for (L/S), Generator for (L/M)): (b) (Q2, Q2), (c) (Q2, Q4), (d) (Q4, Q2), (e) (Q4, Q4).

VI. Conclusion

In this paper, we proposed a dual-purpose integer division algorithm for both offline and online division. The algorithm uses interval analysis to find quotient digits from offline or online operands. The algorithm uses the conventional binary number system. The online dividers achieved shorter execution time than the offline dividers for several online division cases at the cost of more energy consumption and area. We analyzed the characteristics of the dividers and found relationships among the properties of the operands, dependency types, and execution times from the simulation results.

Acknowledgment

This work was supported by the Defense Advanced Research Projects Agency (DARPA) Young Faculty Award under Grant D16AP00119.

References

[1] J. Seo and D. H. Kim, "Dual-Purpose Hardware Algorithms and Architectures – Part 1: Floating-Point Division," in *Proc. IEEE Int. Symp. on Computer Arithmetic*, 2023.

[2] A. Nannarelli, "Performance/Power Space Exploration for Binary64 Division Units," in *IEEE Trans. on Computers*, vol. 65, no. 5, May 2016, pp. 1671–1677.

[3] S. Amanollahi and G. Jaberipur, "Energy-Efficient VLSI Realization of Binary64 Division with Redundant Number Systems," in *IEEE Trans. on VLSI Systems*, vol. 25, no. 3, Mar. 2017, pp. 954–961.

[4] J. D. Bruguera, "Low Latency Floating-Point Division and Square Root Unit," in *IEEE Trans. on Computers*, vol. 69, no. 2, Feb. 2020, pp. 274–287.

[5] F. Lyu, Y. Xia, Y. Chen, Y. Wang, Y. Luo *et al.*, "High-Throughput Low-Latency Pipelined Divider for Single-Precision Floating-Point Numbers," in *IEEE Trans. on VLSI Systems*, vol. 30, no. 4, Apr. 2022, pp. 544–548.

[6] J. D. Bruguera, "Low-Latency and High-Bandwidth Pipelined Radix-64 Division and Square Root Unit," in *Proc. IEEE Int. Symp. on Computer Arithmetic*, 2022, pp. 10–17.

[7] K. Trivedi and M. Ercegovac, "On-Line Algorithms for Division and Multiplication," in *IEEE Trans. on Computers*, vol. C-26, no. 7, Jul. 1977, pp. 681–687.

[8] K. Trivedi, "Higher Radix On-Line Division," in *Proc. IEEE Int. Symp. on Computer Arithmetic*, 1978, pp. 164–174.

[9] O. Watanuki and M. Ercegovac, "Floating-Point On-Line Arithmetic: Algorithms," in *Proc. IEEE Int. Symp. on Computer Arithmetic*, 1981, pp. 81–86.

[10] O. Watanuki and M. Ercegovac, "Floating-Point On-Line Arithmetic: Error Analysis," in *Proc. IEEE Int. Symp. on Computer Arithmetic*, 1981, pp. 87–91.

[11] M. Ercegovac, "On-Line Arithmetic: An Overview," in *Real Time Signal Processing VII: Proc. SPIE*, vol. 495, 1984, pp. 86–93.

[12] P. K.-G. Tu and M. D. Ercegovac, "A Radix-4 On-Line Division Algorithm," in *Proc. IEEE Int. Symp. on Computer Arithmetic*, 1987, pp. 181–187.

[13] A. F. Tenca, A. Shantilal, and M. Sinky, "A Radix-4 On-line Division Design and Its Application to Networks of On-line Modules," in *Proceedings of SPIE*, vol. 5205, 2003, pp. 529–540.

[14] M. Ercegovac and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementation*. Kluwer Academic Publishers, 1994.

[15] N. Takagi, S. Kadowaki, and K. Takagi, "A Hardware Algorithm for Integer Division," in *Proc. IEEE Int. Symp. on Computer Arithmetic*, 2005, pp. 1–7.