An Energy-Efficient Approximate Divider Based on Logarithmic Conversion and Piecewise Constant Approximation

Yong Wu, Honglan Jiang, *Member, IEEE*, Zining Ma, Pengfei Gou, Yong Lu, Jie Han, *Senior Member, IEEE*, Shouyi Yin, *Member, IEEE*, Shaojun Wei, *Fellow, IEEE*, and Leibo Liu, *Senior Member, IEEE*

***Abstract*—Approximate computing (AC) has been considered as a promising paradigm to improve the energy-efficiency of computing hardware for error-tolerant applications, with negligible quality degradation to the output. Dividers frequently limit the performance of a computing system; however, they have not received as much attention as multipliers and adders in AC. In this paper, an energy-efficient and high-performance approximate divider is proposed based on logarithmic conversion and piecewise constant approximation. In this design, the range for the conversion between binary and logarithmic numbers is first expanded from $[0,1]$ to $[-0.5,1]$. A heuristic search algorithm is then devised to find the most accurate constant set to approximate the reciprocal of the divisor, by minimizing a statistical error. The hardware implementation is presented for both floating-point (FP) and integer dividers. With a high configurability, the proposed divider results in a mean relative error distance (MRED) from 2.78% to 0.046%, indicating a high accuracy among state-of-the-art approximate dividers. Compared to the half-precision FP divider, the proposed divider with a MRED of 0.74% can achieve nearly 90× improvement in PDP. Moreover, compared to state-of-the-art approximate dividers, the proposed design is in the Pareto Frontier in terms of power delay product (PDP) and MRED. The three image processing application results demonstrate that the proposed divider can result in the highest peak signal-to-noise ratio (PSNR) and structural similarity (SSIM) even with truncation.**

***Index Terms*—Approximate computing, divider, logarithmic conversion, heuristic constant-set searching.**

I. Introduction

With the unprecedented development of big data and artificial intelligence (AI) technologies, Giga or even Tera bits of data needs to be processed within a second. However, the improvement in hardware-efficiency due to the scaling of CMOS transistors is becoming very difficult and costly as the feature dimension reaches to nanometer level [1]. Other traditional approaches to improving performance including parallelization and pipelining take extra area and frequently consume more power. Power dissipation has become the major obstacle to improve computing performance across most technology platforms. Thus, exploring new computing paradigms with high performance and energy efficiency is imperative.

Many contemporary computing-intensive applications, such as face detection [2] and speech recognition [3], can inherently tolerant some errors without introducing severe quality degradation to the output [4], [5]. Due to the limitation of sampling, transmission and quantization techniques, the inputs to be processed in the digital system are not exactly the same as the natural signals. Some applications are sensitive only to statistical results; thus, the positive and negative errors can eliminate each other in accumulation operations [6], [7]. In addition, human perception can hardly distinguish minor differences between the accurate and approximate results with controlled errors [8], [9]. Therefore, pursuing completely accurate results at the cost of sacrificing the hardware capability is not always necessary.

Consequently, approximate computing (AC) has emerged to obtain high-performance and energy-efficient hardware with a slight quality degradation in the final results [2], [10]–[12]. A large number of work has been focused on the approximation of arithmetic circuits, including adders, multipliers and dividers [2]. Moreover, [13] has demonstrated approximate computing for keyword-spotting as a pioneer silicon prototype.

Approximate adders and multipliers have been extensively studied during the last decade [14]–[17]. In contrast, approximate dividers have not received much attention until recent years. Some effort has been made on the approximation of the exact integer dividers via circuit simplification [18]–[20]. Chen et al., designed three types of approximate subtraction modules at the transistor level to substitute the exact ones in unsigned non-restoring and restoring dividers [18], [19]. To pursue a higher processing speed, an approximate adder is designed for a high-radix array divider with more energy dissipation than the radix-2 design [21]. In [22], several most significant bits (MSB) are selected and processed by using a reduced-width divider. A 12/6 exact divider is required to limit the mean relative error distance (MRED) within 1%

This work is supported by the National Natural Science Foundation of China under Grant 62104127, the National Natural Science Foundation of China under Grant 61834002, the National Key R&D Program of China under Grant 2018YFB2202101 and the National Science and Technology Major Project of the Ministry of Science and Technology of China under Grant 2018ZX01027101-002). (Corresponding authors: Honglan Jiang; Leibo Liu.)

Y. Wu, Z. Ma, S. Yin, S. Wei and L. Liu are with the School of Integrated Circuits, Tsinghua University, Beijing, 100084, China.
E-mail: {wuyong20, maz20}@mails.tsinghua.edu.cn, {yinsy, wsj, liulb}@tsinghua.edu.cn

H. Jiang is with the Department of Micro-Nano Electronic, Shanghai Jiao Tong University, Shanghai, 200240, China.
E-mail: honglan@sjtu.edu.cn

P. Gou and Y. Lu are with the HeXin Technology Co., Ltd., Beijing-Guangzhou Collaborative Innovation Center, Guangzhou, Guangdong, China.
E-mail: goupengfei@shingroup.cn, luyong@shingroup.cn

J. Han is with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada.
E-mail: jhan8@ualberta.ca

for the 16/8 division, which gains little when considering peripheral modules. To avoid the overflow due to the above simple truncation, two pruning schemes have been introduced in the adaptively approximate divider (AAXD) [23], resulting in a higher accuracy but with more hardware resources.

Also, some new techniques at the algorithmic or architectural level are explored to obtain energy-efficient and high-performance approximate dividers. For instance, a quality-configurable approximate divider approximates the reciprocal of the divisor using Taylor series expansion [24]. More than three iterations are required to restrict the MRED within 1%, which introduces a large latency and energy consumption. A truncation-based approximate divider (TrunApp) is designed by directly approximating the reciprocal of the divisor as several binary constants [25]. Thus, the division is implemented by using multiplication. To further simplify the circuit, truncation is applied to the inputs. Due to the direct approximation and the absence of constant selection, the TrunApp is less accurate. Different from the above introduced integer designs, in [26], an energy-efficient FP approximate divider (FPAD) is proposed by using a multistage approximation methodology. It separates the dividend and divisor into several intervals. Each interval is approximated by using a linear function. The total number of intervals increases exponentially with the increase of approximation level. Deep level is needed to obtain a reasonable error, which leads to a large circuit area and power consumption.

A promising type of dividers approximated at algorithmic level are based on the logarithmic number system (LNS). In these designs, an approximate logarithmic algorithm proposed by Mitchell [27] is used to transfer division into subtraction, which can achieve significant hardware improvements especially for FP numbers. However, the basic logarithmic approximate divider (ALD) produces relatively large errors due to a simple conversion between the binary and logarithmic numbers, e.g., over 4% in MRED. In addition, the approximate quotient always overestimates the exact result, resulting in positive errors for all inputs that may be accumulated in subsequent stages. To cope with this issue, an approximate FP divider with near-zero error bias (FaNZeD) has been proposed [28]. Specifically, the error bias in ALD is estimated and compensated by using an 8-bit subtractor. As a result, the error bias and MRED of FaNZeD can be significantly reduced. However, some extra circuits are required to normalize the obtained result into the FP representation, which increases the critical path and decreases the speed of the divider.

As per the above observations, most approximate dividers are designed for processing integer numbers. However, FP divisions can be necessary for some applications requiring a wider dynamic range or higher accuracy. The exact IEEE 754 single-precision FP divider results in more than 15 times the area-delay-product (ADP) of the corresponding FP multiplier [28]. Thus, a high-performance and resource-efficient approximate FP divider is urgently demanded for the error-resilient applications involving division. Taking advantages of the LNS-based architecture, a hardware-efficient approximate divider with high accuracy, mainly for FP inputs, is designed in this paper. In the divider, Mitchell's algorithm is modified to decrease the conversion error between binary and logarithmic numbers. A piecewise constant approximation is then applied to transfer the division into a simple multiplication. Compared to previous work, the contributions of this paper are as follows.

- An extension of logarithmic conversion is proposed for designing a hardware-efficient approximate divider with configurable accuracy.
- A heuristic search algorithm is proposed for the piecewise constant approximation to find the most accurate constants for the reciprocal of the divisor resulting in the minimal statistical error.
- The hardware implementation of the proposed divider is explicitly presented with low complexity and short critical path. The architectures for both FP and integer inputs are presented.
- The proposed divider is compared with state-of-the-art approximate dividers in terms of errors and circuit measurements. The proposed divider has the smallest MRED and is in the Pareto Frontier when considering both the MRED and PDP together.
- The approximate dividers are further assessed in three image processing applications. With a lower hardware consumption, the proposed divider produces the higher output quality than the other approximate designs.

The remainder of this paper is organized as follows. The next section introduces the preliminaries including FP and logarithmic number systems. Section III presents the basic theory of the proposed divider. The hardware architectures for both FP and integer inputs are depicted in Section IV. Section V analyzes the relative error distance theoretically and presents the simulation results in terms of statistical errors. The circuit measurements including power, area and delay are also presented in this section. In Section VI, three image processing applications are used to examine the quality of the proposed and existing dividers. The paper is concluded in Section VII.

II. Preliminaries

A floating-point (FP) number system can process a large range of data with a high precision; thus, it has been used in many areas, such as scientific computing and machine learning. However, FP arithmetic circuits are generally very time- and resource-consuming, which significantly limit the processing speed and increases the power consumption. Logarithmic number system (LNS) can transfer FP multiplication/division operations into addition/subtraction operations, which greatly improves the energy efficiency and operating performance.

A. Floating-point number system

In the FP number system, a number F consists of three parts, the sign S, the exponent E and the mantissa M, as shown in Fig. 1. The S has one bit, where 0 and 1 are used to represent positive and negative numbers, respectively. The lengths of the exponent and mantissa depend on specific applications and can be changed flexibly. For the single-precision (32-bit) representation specified in IEEE 754 standard, they are 8 and 23 bits, respectively, denoted as FP (1, 8, 23). Correspondingly, FP (1, 5, 10) and FP (1, 3, 4) represents standard half-precision

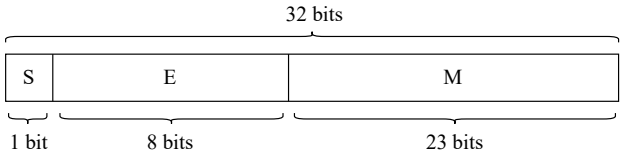


Fig. 1: The standard 32-bit floating-point number representation.

(16-bit) and 8-bit FP numbers. An FP number based on binary is expressed as

$$F = (-1)^S \cdot 2^{E-bias} \cdot (1+M), \tag{1}$$

where the $bias$ represents the skewing of the exponent that ensures it is non-negative. For IEEE 754 standard single-precision representation, the bias is 127 and the actual exponent range is [-126, 127]. The mantissa includes a fractional part denoted as M and a hidden one. The M is within [0,1) and the actual value for the mantissa is in [1,2).

Considering the division operation of two FP numbers, dividend A and divisor B, the quotient Q can be denoted as

$$Q_S = A_S \oplus B_S \tag{2}$$

$$Q_{M1} = \begin{cases} (1+A_M)/(1+B_M) & \text{if} A_M \geq B_M \\ 2(1+A_M)/(1+B_M) & \text{otherwise} \end{cases} \tag{3}$$

$$Q_E = \begin{cases} A_E - B_E + bias & \text{if} A_M \geq B_M \\ A_E - B_E + bias - 1 & \text{otherwise} \end{cases}, \tag{4}$$

where A_S (B_S, Q_S), A_E (B_E, Q_E) and A_M (B_M, Q_M) represent the sign, exponent and mantissa of A (B, Q), respectively. The Q_{M1} combines the quotient mantissa Q_M with its hidden one, and this representation are used in the rest of paper, the symbol $\oplus$ denotes the XOR operation.

In the conventional FP division, a divider, a subtractor, several multiplexers and a normalization module are needed for implementing the hardware architecture. However, overwhelming latency and hardware resource are inevitable due to the exact divider for the mantissa computation. Thus, to improve the hardware efficiency of the FP divider, many methodologies have been attempted by focusing on the simplification of the mantissa divider, such as truncation and linear approximation [22], [25].

B. Logarithmic number system

Logarithmic conversion has been used to simplify the calculation of multiplication and division [27]. In the binary logarithm, the input number must be positive and the sign can be processed separately. A magnitude of a FP number can be transformed to a logarithm number using the following equation

$$\lg F = \log_2 F = E - bias + \lg(1+M). \tag{5}$$

For simplification, we use lg to replace $\log_2$ in this paper. Equation (5) shows that the integer part of a logarithm number is equal to the exponent of the FP number. The key then becomes transforming the mantissa of the FP number to the logarithmic form. Mitchell [27] proposed an approximate method for $M \in [0,1)$ as follows

$$\lg(1+M) \approx M, \tag{6}$$

in this case, the magnitude of an FP number can be simply transformed to the logarithm form by adding its exponent and mantissa, i.e.,

$$\lg F \approx E - bias + M, \tag{7}$$

By adopting this approximation, (3) can be converted to

$$Q^l_{M1} = \begin{cases} A_M - B_M & \text{if } A_M \geq B_M \\ A_M - B_M + 1 & \text{otherwise} \end{cases}, \tag{8}$$

where Q^l_{M1} is the logarithm form of the mantissa division. The processing of the sign and exponent parts are the same as (2) and (4). Similarly, the anti-logarithm result can be approximately obtained by

$$2^M \approx 1 + M, \tag{9}$$

Thus, the final outcome of (3) can be approximated as

$$Q_{M1} = \begin{cases} A_M - B_M + 1 & \text{if } A_M \geq B_M \\ A_M - B_M + 2 & \text{otherwise} \end{cases}, \tag{10}$$

According to (2), (4) and (10), an approximate divider based on logarithmic conversion can be implemented by using only two subtractors, a normalization module and several necessary components, which greatly decreases the latency and improves the hardware efficiency. However, the error for this approximate divider is relatively large that limits its use in many error-tolerant applications demanding a high accuracy.

III. Proposed approximate divider

In this section, a novel approximate divider based on logarithmic conversion and piecewise constant approximation (LPCAD) is proposed. To alleviate the large errors introduced by the Mitchell's approximation, we extend the approximation scope of (6) from [0,1] to [-0.5,1] and approximate the logarithm function by using two linear functions. A piecewise constant approximation towards the divisor is then employed to transform the division into several additions that can significantly decrease the latency and power consumption.

A. Extension of the logarithmic conversion for division

Considering two FP operands, dividend A and divisor B, the logarithm of the quotient can be expressed as

$$\begin{aligned} Q^l &= A_E - B_E + \lg\frac{1+A_M}{1+B_M} \\ &= A_E - B_E + \lg(1+\frac{A_M - B_M}{1+B_M}) \\ &= A_E - B_E + \lg(1+x), \end{aligned} \tag{11}$$

where, $x = (A_M - B_M)/(1+B_M)$, A_M, $B_M \in [0,1)$. When $A_M \geq B_M$, x is non-negative; otherwise, x is negative. It can be proven that $x \in (-0.5, 1)$ and cannot be directly approximated

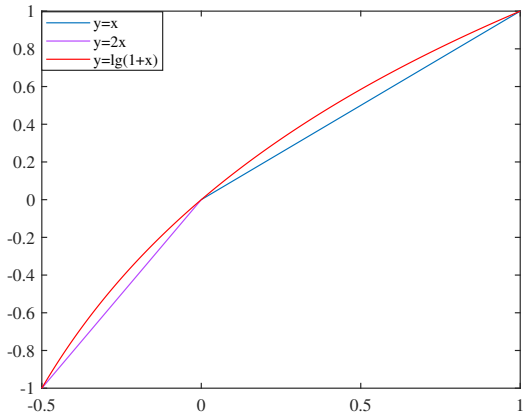


Fig. 2: Approximate conversion from $\log_2(1+x)$ to linear functions for $x \in [-0.5, 1]$

.

by using (6). Thus, we extend the scope of M in (6) from [0,1] to [-0.5,1] as

$$\lg(1+x) = \begin{cases} 2x & \text{if } x \in [-0.5, 0) \\ x & \text{if } x \in [0, 1] \end{cases}. \tag{12}$$

To see the difference between the accurate and approximate results intuitively, their curves are plotted. As shown in Fig. 2, the approximation using (12) always underestimates the exact outcome, leading to error bias that is similar to the Mitchell's approximation. However, this problem can be partially solved by the following approximation.

By using (12), an FP division can be calculated as

$$Q^l_{M1} = \begin{cases} (A_M - B_M)/(1+B_M) & \text{if } A_M \geq B_M \\ 2(A_M - B_M)/(1+B_M) + 1 & \text{otherwise} \end{cases}. \tag{13}$$

The sign and exponent are processed as (2) and (4). It should be noted that $Q^l_{M1} \in [0, 1)$. Thus, (13) can be transformed through (9) as

$$Q_{M1} = \begin{cases} (A_M - B_M)/(1+B_M) + 1 & \text{if } A_M \geq B_M \\ 2(A_M - B_M)/(1+B_M) + 2 & \text{otherwise} \end{cases}. \tag{14}$$

The calculation still requires division operation. Next, a simple but effective method is introduced to approximate the reciprocal of $(1+B_M)$, converting the division into several addition operations.

B. Piecewise constant approximation

To further simplify the calculation, piecewise constant approximation (PCA) methodology is applied to substitute the reciprocal of $(1+B_M)$ with some constants according to the value of B_M. For ease of implementation, the number of binary bits to present the constants should be as few as possible. However, for a high accuracy, a large width is required for the constants. As $1/(1+B_M) \in (0.5, 1]$, if taking the most k significant bits of B_M into account, $(k+1)$-bit constants is used here for a comprehensive consideration of the accuracy and hardware cost. For example, if the most significant two bits of the B_M is considered, the binary set $\{0.100, 0.101, 0.110, 0.111\}$ can be used to approximate the reciprocal of $(1+B_M)$. By using PCA, (14) is approximated as

$$Q_{M1} = \begin{cases} (A_M - B_M) \times B_{appro} + 1 & \text{if } A_M \geq B_M \\ 2(A_M - B_M) \times B_{appro} + 2 & \text{otherwise} \end{cases}, \tag{15}$$

where B_{appro} is the constant set for approximating $1/(1+B_M)$. In each range of B_M, there exists a most appropriate constant to approximate $1/(1+B_M)$.

To automatically select the most appropriate constant for a specific range of B_M, a heuristic search algorithm is developed aiming at minimizing the statistical error. Here, we use the mean relative error distance (MRED) as the goal, which is defined as

$$\text{MRED} = \frac{1}{N} \sum_{i=1}^{N} \left| \frac{O^i_{app} - O^i_{ext}}{O^i_{ext}} \right|, \tag{16}$$

where N is the sample number, O^i_{app} and O^i_{ext} are the approximate and exact outputs, respectively.

Algorithm 1 shows the searching process of the heuristic algorithm for the most appropriate set of constants for each range of B_M. The input arguments are the FP dividends A, divisors B, the total number of inputs n and the number of the MSBs in the mantissa of the divisor to be taken into account for the approximation, k. The aim is to obtain a set of constants O_c that closely approximates $1/(1+B_M)$. The ranges to be studied for the input divisor mantissas are obtained first according to k. Choosing the most suitable constant for every range of divisor mantissa from the candidates is implemented from line 6 to 27. For fairness, in the process of searching the constant for the current range, each other range is approximated by the same constant as shown in line 14 to 17, the current range is approximated by the four candidates as shown in line 19 to 21. Four candidates for each range are generated (line 7); it ensures one candidate larger, and one less than the range boundaries, and two within the range in the worst case according the following constant established condition

$$\frac{1}{1+m/2^k} - \frac{1}{1+(m+1)/2^k} \leq 2^{-k}, \tag{17}$$

where $0 \leq m \leq 2^k - 1, m \in \mathbb{N}^+$, the left denotes the interval of the reciprocals of two continuous k-bit input divisor mantissas. Then, for each candidate, the MRED of the approximate division is computed as shown in line 23 to 25. The candidate that leads to the minimal MRED is selected as the approximate constant for the current range, as shown in line 26 to 27. When the whole loop is finished, the best set of constants is obtained. Also, the optimization goal of this algorithm can be other statistic error metrics such as the normalized mean error distance.

By utilizing Algorithm 1, the sets of constants to approximate $1/(1+B_M)$ for $k=2$ and $k=3$ are listed in Table I. For the majority of ranges, the approximate constants are within the ranges. For some ranges, the constants can be the same, such as $[0.625, 0.750)$ and $[0, 750, 0.875)$ for $k=3$, both of the approximate constants are 0.5625, that is 0.1001 in the

Algorithm 1 The heuristic algorithm for searching the most appropriate constants to approximate the reciprocal.

Input: A_j, B_j; *// the random/uniform inputs,* $j = \{1, \ldots, N\}$. k; *// the number of the most significant bits in the mantissa for the divisor to be considered for the approximation.*
Output: O_c^i; *// constants for each range of the divisor mantissa,* $i = \{1, \ldots, 2^k\}$.
// Initialization
$O_c = 1$; *// initialize the constants to 1s.*
$O_{seg} = zeros$;
for $i = 2$ to $2^k + 1$ **do**
  $O_{seg}^i = O_{seg}^{i-1} + 1/2^k$; *// calculate each interval of inputs.*
end for
for $i = 2$ to $2^k + 1$ **do**
  $temp = [O_{const}^{i-1}, \cdots, O_{const}^{i-1} - 3/2^{k+1}]$; *// four candidates for the approximate constant in each interval.*
  for $r = 1$ to 4 **do**
    if temp(r)$\times O_{seg}^i > 0.5$ **then**
      temp(r)=0; *//To avoid the complex normalization module in the mantissa part.*
    end if
  end for
  for $j = 1$ to n **do**
    for $l = 1$ to 2^k **do**
      if $O_{seg}^{l-1} \le B_M^i < O_{seg}^l$ **then** *// determine which range the* j*th divisor mantissa belongs to..*
        $B_{appro}^j(1:4) = O_c^{l+1}$; *//allocate four same approximate constants for each divisor.*
      end if
    end for
    if $O_{seg}^{i-1} \le B_M^j < O_{seg}^i$ **then** *//determine whether the divisor mantissa is in the current search range.*
      $B_{appro}^j(1:4) = temp$; *//allocate four different candidates for the divisor mantissa in the current search range.*
    end if
  end for
  $Q_e = Divider\_Ext(A, B)$; *//calculate the exact quotients.*
  $Q_a = Divider\_Appro(A, B, B_{appro})$; *//calculate four approximate quotients.*
  $V_{MRED} = MRED(Q_e, Q_a)$; *//calculate four MREDs.*
  $[V, index] = min(V_{MRED})$; *//search the minimal MRED and obtain the corresponding index for temp.*
  $O_c^i = temp(index)$; *//determine the most appropriate constant for the current mantissa range.*
end for
O_c; *// output the most appropriate set of constants that minimize the MRED.*

binary format. For $k = 2$, (15) can be transformed to (18) when $A_M \ge B_M$ and (19) when $A_M < B_M$.

$$Q_{M1} = \begin{cases} 0.875(A_M - B_M) + 1 & \text{if } 0 \le B_M < 0.25 \\ 0.75(A_M - B_M) + 1 & \text{if } 0.25 \le B_M < 0.5 \\ 0.625(A_M - B_M) + 1 & \text{if } 0.5 \le B_M < 0.75 \\ 0.5(A_M - B_M) + 1 & \text{if } 0.75 \le B_M < 1 \end{cases}, \quad (18)$$

TABLE I: The set of constants to approximate the reciprocal of $1 + B_m$ for $k = 2$ and $k = 3$.

$k=2$			$k=3$		
B_m	$1/(1+B_m)$	B_{appro}	B_m	$1/(1+B_m)$	B_{appro}
[0.00, 0.25)	(0.80, 1.00]	0.875	[0.000, 0.125)	(0.89, 1.00]	0.9375
[0.25, 0.50)	(0.67, 0.80]	0.75	[0.125, 0, 250)	(0.80, 0.89]	0.875
[0.50, 0.75)	(0.57, 0.67]	0.625	[0.250, 0.375)	(0.73, 0.80]	0.75
[0.75, 1.00)	(0.50, 0.57]	0.5	[0.375, 0.500)	(0.67, 0.73]	0.6875
			[0.500, 0.625)	(0.62, 0.67]	0.625
			[0.625, 0.750)	(0.57, 0.62]	0.5625
			[0.750, 0.875)	(0.53, 0.57]	0.5625
			[0.875, 1.000)	(0.50, 0.53]	0.5

$$Q_{M1} = \begin{cases} 2\times 0.875(A_M - B_M) + 2 & \text{if } 0 \le B_M < 0.25 \\ 2\times 0.75(A_M - B_M) + 2 & \text{if } 0.25 \le B_M < 0.5 \\ 2\times 0.625(A_M - B_M) + 2 & \text{if } 0.5 \le B_M < 0.75 \\ 2\times 0.5(A_M - B_M) + 2 & \text{if } 0.75 \le B_M < 1 \end{cases}. \quad (19)$$

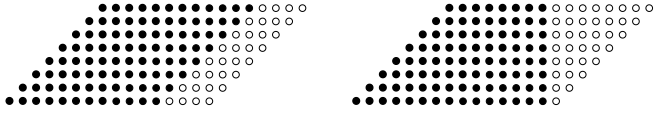
C. Truncation

Truncation is a common technique to reduce the hardware complexity and shorten the critical path [29]. For FP numbers, truncation is more efficient due to the hidden one, especially for multiplication and division. Assuming there is an n bit FP number, we truncate its $n - t$ LSBs and leave its t MSBs to do computation. The largest relative truncation error can be expressed as (20), which decreases exponentially with the increase of t.

$$\text{RED}_{max}^t = 2^{-t} - 2^{-n}, \quad (20)$$

where RED_{max}^t represents the largest relative error distance (RED) introduced by truncation. The worst case occurs when the t MSBs are zeros and the truncated bits are ones.

There are two commonly used truncation methods for designing an approximate multiplier. As shown in Fig. 3(a), the inputs are directly truncated. The advantage for this method is easy to design. However, the error of this method is relatively large. To reduce the error, the other method truncates some least significant partial products with minimal weights, as shown in Fig. 3(b).



(a) truncating inputs (b) truncating partial products

Fig. 3: Two truncation methods. (a) truncating some LSBs of the inputs. (b) truncating some LSBs of partial products.

IV. Hardware implementation

This section introduces the hardware architecture for the proposed approximate divider. The FP design is first introduced, including the architectural and multiplication implementations. To extent the application scope of the proposed division scheme, an architecture for integer inputs is also presented.

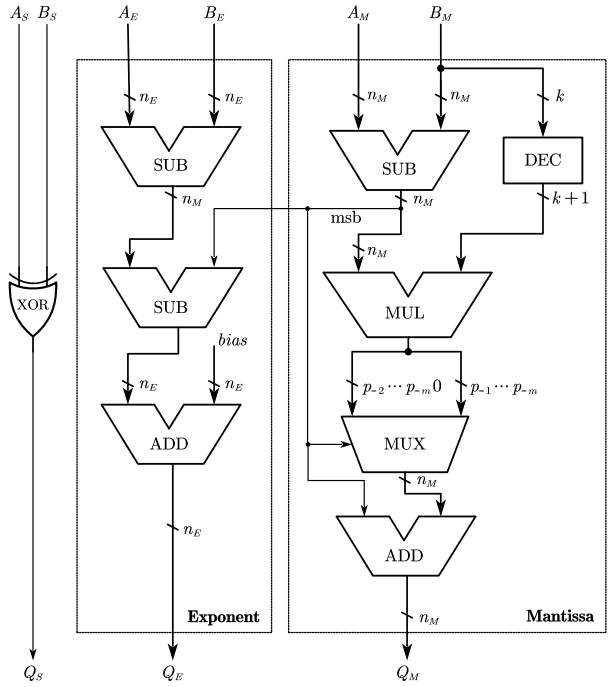


Fig. 4: The LPCAD hardware architecture for FP numbers obtained by directly implementing the (2), (4) and (15).

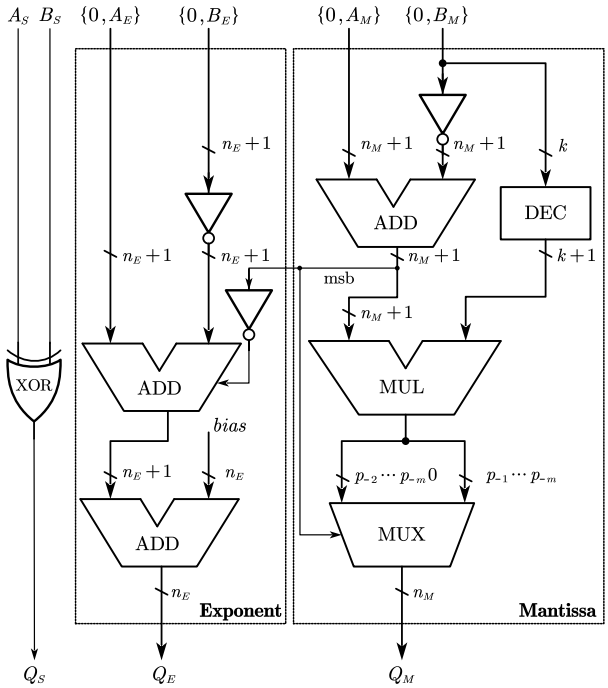


Fig. 5: The proposed LPCAD hardware architecture for FP numbers.

A. LPCAD for FP numbers

As discussed in Sections II and III, the architecture of LPCAD for FP numbers can be derived from (2), (4) and (15). Straightforwardly, the LPCAD can be implemented as Fig. 4, where n_E and n_M are the widths of the exponent A_E (B_E) and mantissa A_M (B_M), respectively. Specifically, the sign of the quotient Q_S is obtained by performing an XOR operation on the signs of the two inputs, A_S and B_S. The exponent Q_E is computed as the subtraction of the dividend and divisor exponents, when A_M is greater than B_M. Otherwise, it needs to be subtracted by one to simplify the normalization process, i.e., ensuring the quotient mantissa Q_M within the range of [0,1). Finally, the bias is added. To calculate the mantissa, B_M is subtracted first by A_M; the subtraction result is then multiplied by the approximate constant B_{appro} generated by using a decoder. If the difference is negative, shift the product left for one bit; otherwise not. A constant 1 is finally added to ensure the output mantissa is in [0,1). Fig. 4 shows the hardware architecture as per the (2), (4) and (15). However, to lower the implementation complexity, Fig. 5 is obtained by considering the calculations from a global perspective.

1) Architectural design: For the exponent calculation, whether subtracting the extra one or not depends on the difference between the dividend and divisor mantissas. We convert the subtraction into the addition of the negative subtrahend obtained by a two's complementary operation. If the MSB of $S = (A_M - B_M)$ is 1, i.e., $A_M < B_M$, the subtraction of 1 can be neutralized by the added 1 due to 2's complementing. Thus, $Q_E = A_E + \overline{B_E}$. Otherwise, $Q_E = A_E + \overline{B_E} + 1$. It can be directly implemented by an adder with a carry-in, where the carry-in is the inverted MSB in S.

Similarly, the implementation for the mantissa computation can be simplified. The subtraction of A_M and B_M is converted to addition by using a two's complementary operation. The product result $P = S \times B_{appro}$ is within (-0.5,1) due to the range limitation constrained by Algorithm 1. Let P be $(p_0.p_{-1}p_{-2}p_{-3}\ldots p_{-M})_2$ in two's complement representation, the two MSBs of the P must both be 1, i.e., 1.1, when $A_M < B_M$. Under this condition, we can implement $\times 2$ by just taking all the right bits of the dot according to (21). Then, an extra 1 is added to the result to generate the final Q_M, as shown in (22). Thus, Q_M is obtained by taking $p_{-2}p_{-3}\ldots p_{-M}$, appended with a 0. If $A_M \geq B_M$, all bits to the right of the dot $(p_{-1}p_{-2}\ldots p_{-M})$ are taken as Q_M. Based on these observations, a multiplexer is needed to select the corresponding bits of the multiplication result for Q_M.

$$2 \times \overline{1}.1p_{-2}p_{-3}p_{-4}\cdots = \overline{1}.p_{-2}p_{-3}p_{-4}\ldots \tag{21}$$

$$\overline{1}.p_{-2}p_{-3}p_{-4}\cdots + 1 = 0.p_{-2}p_{-3}p_{-4}\ldots \tag{22}$$

Consequently, two adders for the exponent generation, an adder, a decoder (for generating B_M related approximate constant), a multiplier and a multiplexer for the mantissa computation, are used for the proposed approximate FP divider. The hardware implementation of the proposed FP approximate divider is depicted in Fig. 5, where the multiplier can be further simplified as follows.

2) Multiplication implementation: In the mantissa computation, instead of using a typical signed multiplier, the multiplication can be specifically implemented. After $C = B_{appro}$ is produced by the decoder, a dedicated designed multiplier is applied to complete $P = S \times C$. The multiplica-

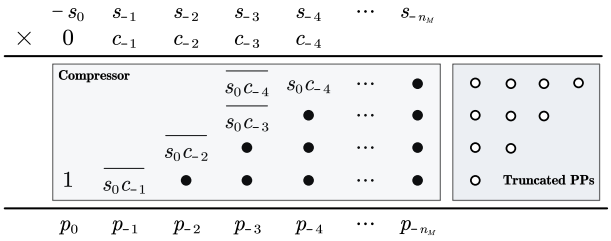


Fig. 6: The dedicated multiplier with truncation.

tion of two signed binary numbers $S = (s_0.s_{-1}\cdots s_{-n_m})_2$ and $C = (c_0.c_{-1}\cdots c_{-(k+1)})_2$ can be expressed as

$$\begin{aligned} P &= S \times C \\ &= (-s_0 + \sum_{i=1}^{n_M} s_{-i}2^{-i}) \times (-c_0 + \sum_{j=1}^{k+1} c_{-j}2^{-j}) \\ &= s_0c_0 - c_0\sum_{i=1}^{n_M} s_{-i}2^{-i} - s_0\sum_{j=1}^{k+1} c_{-j}2^{-j} \\ &+ \sum_{i=1}^{n_M}\sum_{j=1}^{k+1} s_{-i}c_{-j}2^{-(i+j)}. \end{aligned} \tag{23}$$

In the design, the MSB of C is zero. Equation (23) can be simplified as

$$P = -s_0\sum_{j=1}^{k+1} c_{-j}2^{-j} + \sum_{j=1}^{k+1}\sum_{i=1}^{n_M} s_{-i}c_{-j}2^{-(i+j)}. \tag{24}$$

The subtraction in (24) can be implemented by adding the two's complement of $-s_0\sum_{j=1}^{k+1} c_{-j}2^{-j}$ that is given by

$$\begin{aligned} &-s_0\sum_{j=1}^{k+1} c_{-j}2^{-j} = 1 + \sum_{j=1}^{k+1}\overline{s_0c_{-j}}2^{-j} + 2^{-(k+1)} \\ &= 1 + \sum_{j=1}^{k}\overline{s_0c_{-j}}2^{-j} + \overline{s_0c_{-(k+1)}}2^{-k} + s_0c_{-(k+1)}2^{-(k+1)}. \end{aligned} \tag{25}$$

Thus, for $k = 3$, the partial product array for the signed multiplication in (23) is neat, as shown in Fig. 6. Finally, a simple Wallace tree or adder tree with truncation can be adapted to accumulate the partial products.

B. LPCAD for integer numbers

The proposed approximate division scheme can be adapted to implement an integer divider; the unsigned integer divider is introduced here as an example. An unsigned integer includes the exponent and mantissa information in a binary number. Thus, extra components should be added to obtain the exponent and mantissa before inputting to the FP LPCAD. Also, another conversion module is connected to the output of the FP LPCAD to transfer the result in FP format to an integer. Apparently, these additional modules violate the advantages of the proposed approximation scheme compared to the FP design.

Specifically, a leading one position detector (LOPD) and barrel shifter are utilized to transfer the inputs from the integer representation to the FP format [28]. The LOPD generates the exponent E by finding the position of the most significant 1 in the integer input. The barrel shifter is utilized to shift the integer left by E bits and generate the mantissas. For instance, in an 8-bit integer 00101100, the LOPD outputs 101 as the exponent E, and the barrel shifter generates 1.011 as the actual mantissa M_1. The exponent and mantissa are then sent to the LPCAD for FP numbers to calculate the division in FP format. Finally, a barrel shifter is utilized to transfer the FP number into the integer. The LPCAD architecture for integer inputs is depicted in Fig 7.

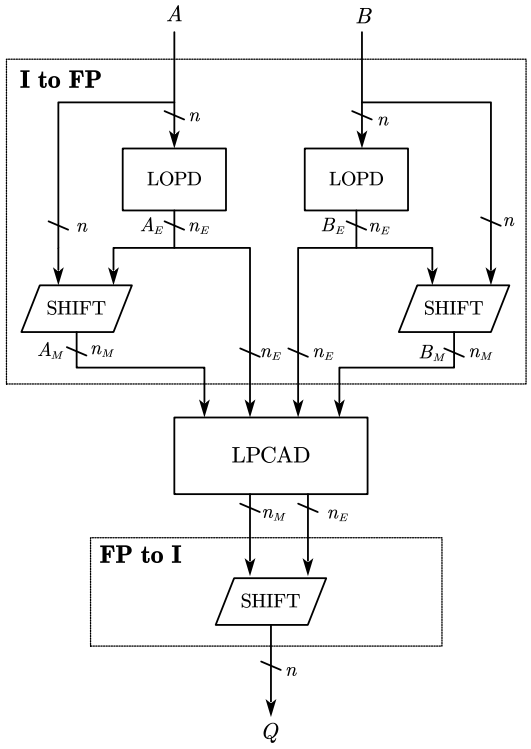


Fig. 7: The LPCAD-based architecture for unsigned integer divider.

V. Error analysis and hardware evaluation

In this section, an error analysis for relative error distance (RED) and simulation results for statistical error metrics, as well as synthesis results for circuit measurements are presented and discussed. In addition, the comparisons with the exact divider and several state-of-the-art approximate dividers are explored.

A. Error analysis

Compared to the exact design, the proposed approximate division scheme generates errors only in the mantissa computation. Thus, the RED without truncation can be expressed as

$$RED = \frac{(A_M - B_M)[B_{appro}(1+B_M)-1]}{1+A_M}, \tag{26}$$

According to (26), both $(A_M - B_M)$ and $[B_{appro}(1+B_M)-1]$ can be positive and negative. Thus, the errors of the approximate division are double-sided and can eliminate each other in

an accumulation operation. Assume the input mantissas A_M and B_M are mutually independent, it can be derived that the maximal absolute RED happens when A_M equals to 0 or approaches to 1 according to (26). The absolute RED satisfies

$$|RED| \leq max\{|B_M[(1+B_M)B_{appro}-1]|, \\ |0.5(1-B_M)[(1+B_M)B_{appro}-1]|\}. \quad (27)$$

It can be noted that the more accurate B_{appro} approximates the reciprocal of $(1+B_M)$, the closer $(1+B_M)B_{appro}-1$ is to 0. Thus, k determines the accuracy of the proposed approximate divider.

As truncation introduces relatively small errors for FP inputs, truncation is also utilized to further reduce the hardware cost especially for a large k. We use t to represent the residual number of bits in input mantissas A_M and B_M. Similarly, the RED with truncation is calculated by (28).

$$RED_t = \frac{B_{appro}(1+B_M)(\lfloor A_M \rfloor_t - \lfloor B_M \rfloor_t) - (A_M - B_M)}{1+A_M}. \quad (28)$$

B. Simulation results for error assessment

1) The statistical error metrics of FP LPCAD: One million random input combinations are used to calculate the MRED and error bias of the FP LPCAD on MATLAB tool. The error bias denoted by Bias is calculated as

$$\text{Bias} = \frac{1}{N}\sum_{i=1}^{N}\frac{O^i_{app} - O^i_{ext}}{O^i_{ext}} \quad (29)$$

Fig. 8 shows the obtained MRED and Bias results for the 16-bit FP LPCAD with an ascending k. With the increase of k, the approximation for $1/(1+B_m)$ is closer to its accurate value and thus, the MRED of the FP LPCAD is decreased. When increasing k from 1 to 7, the MRED of the FP LPCAD decreases from 2.78% to 0.046%. The MRED is below 1% as long as k is larger than 2. The Bias of the FP LPCAD is always less than 0.2% regardless of the value of k. When $k=3$, the Bias reaches the maximal value of 0.20%. As the hardware cost for the decoding and multiplication parts in LPCAD increases with k, to trade-off between the hardware overhead and accuracy, the designs with $k=2$ and $k=3$ are preferred and mainly discussed in this paper. For some applications requiring a higher accuracy, LPCAD with larger k can be adopted. Fig 9 shows the MRED and Bias with respect to the truncation parameter t for the FP LPCADs with $k=2$ and $k=3$, where t is the number of remained columns in the partial product array for compression. As per Fig 9, the MRED and Bias generally decrease with the increase of t, except for the Bias when $k=3$. The MRED is approaching to a minimal constant as t is larger than 8 for both $k=2$ and $k=3$. For the Bias, it is less than 0.1% when $t>7$ for $k=2$ and less than 0.2% when $t>6$ for $k=3$. To trade off the accuracy and hardware cost, LPCAD (2, 8) and LPCAD (3, 8) are mainly considered, where LPCAD (k,t) denotes the LPCAD selecting k bits of B_M for the approximate constant generation and compressing t most significant columns of partial product in the multiplication.

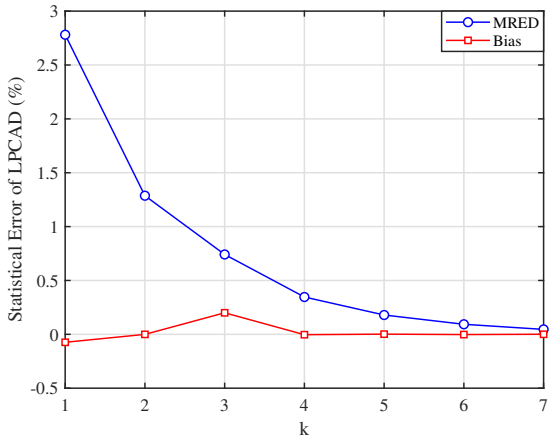


Fig. 8: The MRED and Bias of 16-bit FP LPCAD with different k values.

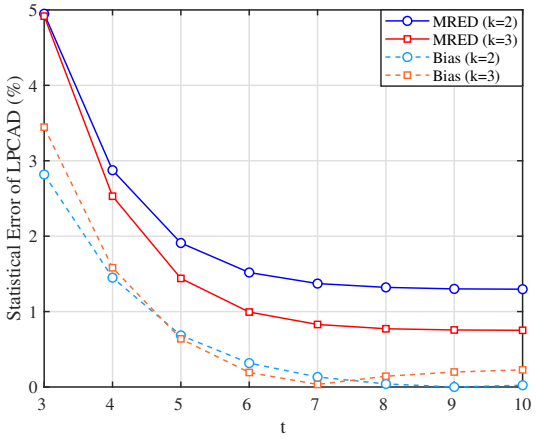


Fig. 9: The MRED and Bias of 16-bit FP LPCAD with different t values when $k=2$ and $k=3$.

Table II shows the MRED and Bias results for the 8-, 16- and 32-bit FP LPCADs with $k=2$ and $k=3$. For 32-bit FP LPCAD, the MRED has a similar characteristic to the 16-bit design, i.e., when the t is equal to or larger than 8, the MRED is approaching to a constant, 1.3% for $k=2$ and 0.74% for $k=3$. Thus, LPCAD (2, 8) and LPCAD (3, 8) are superior for 32-bit FP LPCAD when considering the accuracy and hardware cost. For the 8-bit FP LPCAD, the MRED is always decreasing when t increases from 1 to 5 for both $k=2$ and $k=3$. Thus, truncation is not efficient for an 8-bit FP LPCAD. However, the final choice of LPCAD with different k and t depends on the accuracy requirement of specific error-tolerant applications.

2) Comparison with state-of-the-art approximate dividers: The proposed FP LPCAD is compared with four state-of-the-art approximate divider designs, including two FP dividers, FaNZeD [28] and FPAD [26], two designs that using approximate integer dividers for the mantissa division, TrunApp [25] and AAXD [23]. In addition, the original logarithmic design, ALD [27], is considered as a baseline. Using the same methodology as the FP LPCAD, the MRED and Bias for each divider

TABLE II: Comparison of the MRED and Bias for the FP dividers with three bit-widths.

8-bit FP dividers					
Designs	MRED (%)	Bias (%)	Designs	MRED (%)	Bias (%)
LPCAD (2, 4)	2.83	0.00	ALD (5)	5.63	5.63
LPCAD (2, 5)	1.98	0.79	TrunApp4 (4)	4.74	3.04
LPCAD (3, 4)	2.17	-0.29	FaNZeD (5)	2.80	0.06
LPCAD (3 ,5)	1.39	0.72	FPAD33 (5)	3.47	2.54
			FPAD43 (5)	2.37	-0.08
			AAXD (4)	14.07	8.26
16-bit FP dividers					
Designs	MRED (%)	Bias (%)	Designs	MRED (%)	Bias (%)
LPCAD (2, 4)	2.87	-1.45	ALD (10)	4.07	4.07
LPCAD (2, 8)	1.32	-0.04	TrunApp4 (4)	4.03	1.44
LPCAD (2, 10)	1.30	0.02	FaNZeD (10)	2.71	0.06
LPCAD (3, 4)	2.53	-1.58	FPAD33 (10)	3.52	2.74
LPCAD (3, 8)	0.77	0.14	FPAD43 (10)	2.16	0.39
LPCAD (3, 10)	0.75	0.23	AAXD (10)	1.59	0.71
32-bit FP dividers					
Designs	MRED (%)	Bias (%)	Designs	MRED (%)	Bias (%)
LPCAD (2, 4)	2.88	-1.49	ALD (23)	4.07	4.07
LPCAD (2, 8)	1.31	-0.09	TrunApp4 (4)	4.02	1.40
LPCAD (2, 23)	1.28	0.00	FaNZeD (23)	2.71	0.00
LPCAD (3, 4)	2.54	-1.63	FPAD33 (23)	3.52	2.74
LPCAD (3, 8)	0.77	0.09	FPAD43 (23)	2.17	0.41
LPCAD (3, 23)	0.74	0.20	AAXD (12)	0.79	0.32

are obtained. For TrunApp, the TrunApp4 is considered as it shows the smallest MRED among all series. For FPAD, the FPAD33 and FPAD43 denote the designs with accuracy levels of three and four using two adders, selected by considering their high accuracy and reasonable hardware overhead. The MRED and Bias for the compared dividers are shown on the right side of Table II. Here, ALD (t), TrunApp4 (t), FaNZeD (t), FPAD33 (t) and FPAD43 (t) represent the corresponding divider with a truncation parameter t. AAXD ($2t$) means an exact $2t/t$ divider is utilized in this design. Considering the hardware efficiency, the largest exact divider is limited to 12/6.

As per Table II, when $t > 4$, the MREDs of LPCAD are smaller than any other dividers for both $k = 2$ and $k = 3$ except for the AAXD (12). The MRED of AAXD (12) is close to LPCAD (3,8); however, the required exact 12/6 leads to a higher hardware overhead, as shown in Section V-C. Among the other designs, FPAD43 has the smallest MREDs that are still larger than 2%. The MRED of LPCAD for $k = 2$ and $k = 3$ can be as low as 1.28% and 0.74%, respectively, whereas the minimal MREDs of TrunApp4 and ALD are larger than 4%, and they are beyond 3.4% for FPAD33 and 2.7% for FaNZeD. As for Bias, the results for the LPCAD are lower than 1% for both $k = 2$ and $k = 3$ when $t > 4$ and below 0.1% in some cases. The Bias of LPCAD is slightly larger than that of the FaNZeD because FaNZeD is dedicated designed to limit its Bias to near-zero. As the errors in ALD are single-sided, ALD results in a relatively large Bias. To sum up, the MRED of the proposed LPCAD is remarkably smaller than most approximate dividers and similar to AAXD. Moreover, the Bias of LPCAD with some fixed parameters can be close to zero. This indicates that the LPCAD can be applied to some complex applications with recursive additions where the errors can cancel each other.

C. Hardware evaluation

The considered dividers in Section V-B are implemented in Verilog HDL and synthesized by using the Synopsys Design Compiler based on the TSMC 65-nm standard cell library. The clock periods are set to 30 ns for all evaluated approximate dividers and 50 ns for the exact half-precision FP divider (referred to as HPD), which can ensure no timing violations occur. The critical path delay, area, power dissipation, power-delay-product (PDP) for each design are then obtained. To consider both the hardware efficiency and statistical error, a comprehensive metric APD(E^2) is further calculated, which is defined as

$$\mathrm{APD}(E^2) = \mathrm{Area} \times \mathrm{Power} \times \mathrm{Delay} \times \mathrm{Error}^2. \tag{30}$$

In the work, the MRED is selected as the statistical error because it is a more general metric than the Bias. Besides, the Bias can be zero for some designs, which makes the APDE^2 meaningless.

The synthesis results for 16-bit FP dividers are listed in Table III. The LPCAD (2, 10) and LPCAD (3, 10) can achieve nearly 110× and 90× improvements in terms of PDP when compared to HPD. The HPD is implemented by using a division sign; thus, the HPD is chosen from the DesignWare library as per the synthesis conditions. For the LPCAD, increasing k and t leads to the increase in area, power and delay, but decrease in MRED. Among these approximate dividers, ALD occupies the smallest area because it directly transfers the division into the subtraction operation; for the same reason, it has the largest error especially for Bias. Considering the designs with MREDs between 2% and 3%, LPCAD (2, 4) and LPCAD (3, 4) shows smaller area, power and delay than FaNZeD (10) and FPAD43 (10). Also, LPCAD (2, 8) and LPCAD (2, 10) outperforms AAXD (10) in all circuit measurements, with a smaller MRED. For a same t, the PDP of LPCAD with $k = 2$ is smaller than those of TrunApp4, FPAD33 and FPAD43, whereas the MRED of former is apparently smaller. As for APD(E^2), the LPCAD (3, 4) and LPCAD (3, 8) outperform all state-of-the-art designs. The proposed design is superior to most approximate designs in terms of APD(E^2). To give a clear view over the relationship between the statistical errors and circuit measurements, Fig 10 shows the MRED and PDP of 16-bit approximate dividers together. It can be found that LPCADs forms the Pareto Frontier, which means that the proposed divider outperforms the other approximate dividers in terms of PDP when the same MRED is required.

Also, the synthesis results for 8-bit and 32-bit FP dividers are listed in Table IV. For 32-bit dividers, LPCAD takes more area compared to ALD and FaNZeD at most cases. However, the parameter t affects the MRED little when it decreases from 23 to 8; thus, the LPCAD (2, 8) or LPCAD (3, 8) can be selected for the 32-bit divider. LPCAD (2, 8) and LPCAD (3, 8) show similar area results to ALD (23) and FaNZeD (23) respectively, whereas the former has a smaller MRED, especially for LPCAD (3, 8). The power consumption has a similar tendency as the area. The delay of LPCAD with $k = 3$ is smaller than most designs, except for ALD. For 8-bit dividers, there is no need to truncate due to its significant influence on accuracy and little benefit on hardware performance. As for APD(E^2), the proposed approximate divider also outperforms the other dividers in most cases. Fig 11 reveals the relationship

TABLE III: The circuit and error measurements for 16-bit FP dividers.

Designs	Area (μm^2)	Power (μW)	Delay (ns)	PDP (pJ)	MRED (%)	APD(E^2)
LPCAD (2, 4)	136.8	114.8	2.92	0.335	2.87	377.7
LPCAD (2, 8)	289.2	302.7	3.35	1.014	1.32	511.0
LPCAD (2, 10)	363.2	391.8	3.60	1.410	1.30	865.8
LPCAD (3, 4)	151.2	116.9	2.12	0.248	2.53	239.9
LPCAD (3, 8)	355.2	352.2	3.55	1.250	0.77	263.3
LPCAD (3, 10)	442.4	461.2	3.82	1.761	0.75	438.4
ALD (4)	110.4	75.0	1.69	0.127	4.44	275.9
ALD (10)	175.2	121.0	2.37	0.287	4.07	832.3
TrunApp4 (4)	189.6	130.6	3.43	0.448	4.03	1,379.4
TrunApp4 (10)	413.6	328.5	4.32	1.419	4.95	14,381.7
FaNZeD (4)	123.6	99.2	2.40	0.238	3.88	443.0
FaNZeD (10)	220.4	191.2	4.77	0.912	2.71	1,476.2
FPAD33 (4)	202.0	159.3	2.23	0.355	3.29	776.7
FPAD33 (10)	399.2	351.0	5.03	1.766	3.52	8,732.7
FPAD43 (4)	194.4	156.3	2.77	0.433	3.76	1,189.9
FPAD43 (10)	574.8	486.8	4.01	1.952	2.16	5,235.0
AAXD (8)	262.4	271.5	9.82	2.666	3.21	7,208.7
AAXD (10)	387.6	457.2	15.43	7.054	1.59	6,912.7
HPD (16)	1,807.2	3,792.3	41.10	155.8	0	0

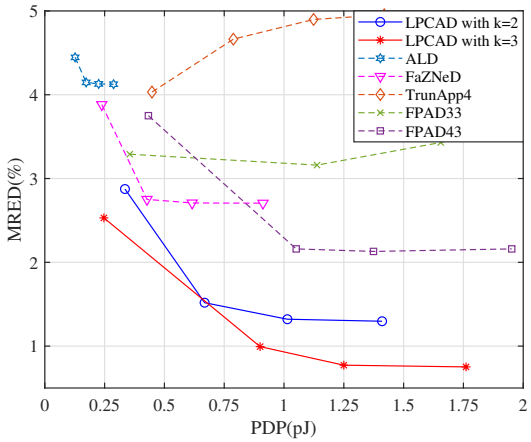


Fig. 10: Comparison of the PDP and MRED for the approximate 16-bit FP dividers. The parameter t is 4, 6, 8 and 10 for all designs, from left to right.

between MRED and PDP of the 32-bit approximate dividers. Likewise, the proposed dividers form the Pareto Frontier.

To sum up, the proposed divider outperforms the other approximate dividers considered in this work in terms of hardware cost when the same MRED is needed. For 16-bit or 32-bit dividers, the LPCAD (2, 8) and LPCAD (3, 8) are selected when considering both the hardware efficiency and accuracy. For 8-bit dividers, the LPCAD (2, 5) and LPCAD (3, 5) with no truncation are advisable choices.

VI. Image Processing Applications

To assess the accuracy of the proposed divider and compare it with the existing approximate dividers in error-tolerant applications, three image processing applications are considered, change detection and foreground extraction using 16-bit FP dividers, K-Means for color quantization using 32-bit FP dividers. To show the differences between the exact and approximate divisions and evaluate the accuracy of the dividers in these applications, the peak signal-to-noise ratio (PSNR) (based on the mean squared error) and structural similarity (SSIM) of the resultant images are calculated.

TABLE IV: Comparison of the circuit and error measurements for 8-bit and 32-bit FP dividers.

8-bit FP dividers						
Designs	Area (μm^2)	Power (μW)	Delay (ns)	PDP (pJ)	MRED (%)	APD(E^2)
LPCAD (2, 4)	124.4	110.9	3.02	0.335	2.83	333.7
LPCAD (2, 5)	167.6	160.9	2.40	0.386	1.98	253.7
LPCAD (3, 4)	138.8	112.2	2.12	0.237	2.17	155.5
LPCAD (3, 5)	204.4	184.5	3.10	0.572	1.39	225.9
ALD (5)	96.8	72.2	2.52	0.182	5.62	556.3
TrunApp4 (4)	162.4	112.2	3.22	0.361	4.73	1,312.7
FaNZeD (5)	112.4	94.5	2.74	0.259	2.80	228.2
FPAD33 (5)	259.2	185.8	4.02	0.747	3.47	2,331.1
FPAD43 (5)	295.2	218.1	3.55	0.774	2.37	1,283.8
AAXD (8)	120.4	104.9	4.26	0.447	3.25	568.3
32-bit FP dividers						
Designs	Area (μm^2)	Power (μW)	Delay (ns)	PDP (pJ)	MRED (%)	APD(E^2)
LPCAD (2, 4)	169.2	138.1	2.92	0.403	2.88	565.9
LPCAD (2, 8)	321.6	326.2	3.35	1.092	1.31	603.1
LPCAD (2, 12)	468.4	505.4	3.81	1.925	1.28	1,477.7
LPCAD (3, 4)	183.6	139.8	2.12	0.296	2.54	351.1
LPCAD (3, 8)	387.6	378.5	3.55	1.343	0.77	308.8
LPCAD (3, 12)	564.8	593.3	4.18	2.480	0.74	767.0
ALD (4)	154.8	107.9	2.03	0.219	4.40	656.4
ALD (23)	360.0	254.1	4.20	1.067	4.07	6,364.2
TrunApp4 (4)	230.4	154.5	3.76	0.581	4.01	2,152.2
FaNZeD (4)	168.0	132.3	2.75	0.364	3.91	934.4
FaNZeD(23)	477.6	348.8	8.41	2.933	2.71	10,289.0
FPAD33 (4)	242.8	185.6	2.49	0.462	3.52	1,390.3
FPAD33 (23)	933.6	860.6	6.16	5.301	3.31	54,224.9
FPAD43 (4)	195.6	159.4	2.77	0.441	3.79	1,240.6
FPAD43 (23)	1014.4	933.6	6.58	6.143	2.17	29,343.7
AAXD (10)	426.0	471.9	15.47	7.300	1.58	7,763.6
AAXD (12)	563.6	715.5	24.88	17.801	0.79	6,261.6

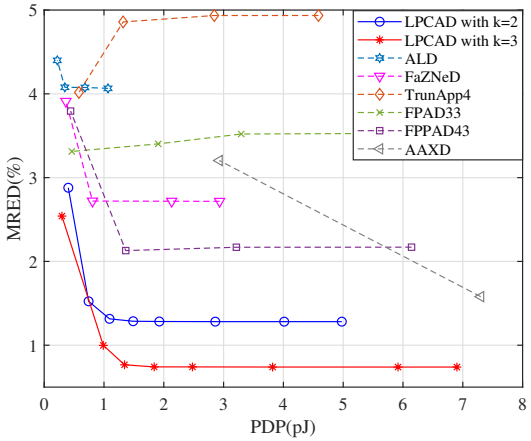


Fig. 11: Comparison of the PDP and MRED for the approximate 32-bit FP dividers. From left to right, the parameter t is 4, 6, 8, 10, 12, 16, 20 and 23 for LPCAD, 8 and 10 for AAXD, and 4, 8, 16 and 23 for the other designs.

A. Change detection

Change detection has been applied to many applications, such as urban planning and disaster assessment, especially when implemented with AI technologies [30]. Change detection is used to distinguish the difference between two input images using the ratios of two image pixels. The outputs

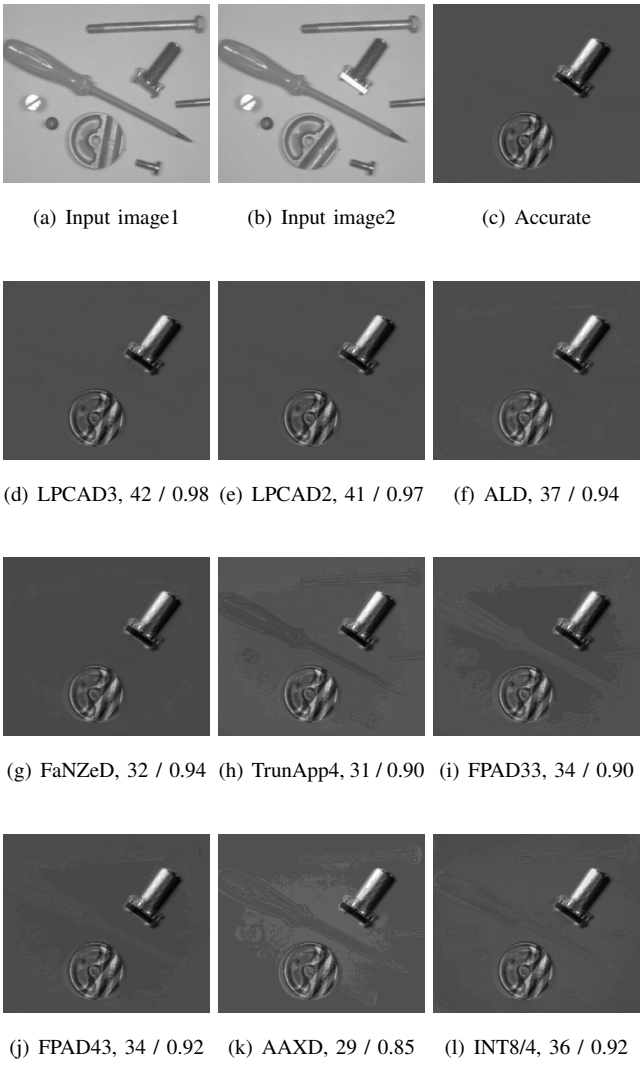
TABLE V: The comparison of the integer and the proposed FP dividers in hardware efficiency and accuracy assessment in change detection.

Dividers	Area (μm^2)	Power (μW)	Delay (ns)	PDP (pJ)	MRED (%)	APD(E^2)	PSNR / SSIM
LPCAD (3, 3)	103.6	71.0	2.00	0.141	4.94	359.0	37 / 0.95
LPCAD (3, 4)	151.2	116.9	2.12	0.248	2.53	239.9	42 / 0.98
LPCAD (3, 6)	267.6	247.3	3.64	0.900	1.00	240.9	46 / 0.98
LPCAD (3, 8)	355.2	352.2	3.55	1.250	0.77	263.3	53 / 0.99
INT (6/3)	112.8	100.9	5.26	0.531	6.52	2,545.0	25 / 0.78
INT (8/4)	232.0	244.5	11.45	2.800	3.21	6,692.4	36 / 0.92
INT (10/5)	396.0	472.3	18.89	8.922	1.59	8,931.8	40 / 0.96
INT (12/6)	608.4	812.0	26.39	21.429	0.79	8,136.5	44 / 0.98

show significant differences in the intensity-change region; otherwise, the outputs tend to be the same to indicate a slight even no change. The division is required for calculating the ratio of pixel values of the original and the target images. If the ratio of two pixels at the same position is far away from 1 (exceeding a certain threshold), the image is considered as changed at this position. In general, 16/8 integer divider is utilized for this application [22]. Thus, when using 16-bit FP dividers, the width of the exponent and mantissa is set to 5 and 10, respectively. Due to its relatively low accuracy requirement, the ALD (4), FaNZeD (4), TrunApp4 (4), FPAD33 (4), FPAD43 (4), AAXD (4), LPCAD (2, 4) and LPCAD (3, 4) are selected to perform the divisions.

Fig 12 presents the output images produced by the considered dividers and the corresponding PSNRs and SSIMs, where the accurate result is obtained by using a double-precision FP divider. Considering PSNR, the LPCAD (3, 4) and LPCAD (2, 4) perform the best, followed by ALD (4), FPAD43 (4), FPAD33 (3), FaNZeD (4), TrunApp4 (4) and AAXD (4). The SSIM results also show that the proposed divider results in a higher image quality in change detection than the other approximate dividers.

To compare the proposed FP divider with the integer divider, several integer divider designs with different sizes (referred to as INT (2t/t)) are also utilized to implement the change detection algorithm. Table V shows the circuit measurements of the dividers, and the PSNRs and SSIMs of the resultant images. It can be seen that, to achieve a similar PSNR or SSIM, the proposed FP divider can be much more hardware-efficient than the integer divider, such as LPCAD (3, 3) and INT (8/4). Note that the synthesized integer dividers are from DesignWare library.



(a) Input image1 (b) Input image2 (c) Accurate

(d) LPCAD3, 42 / 0.98 (e) LPCAD2, 41 / 0.97 (f) ALD, 37 / 0.94

(g) FaNZeD, 32 / 0.94 (h) TrunApp4, 31 / 0.90 (i) FPAD33, 34 / 0.90

(j) FPAD43, 34 / 0.92 (k) AAXD, 29 / 0.85 (l) INT8/4, 36 / 0.92

Fig. 12: Change detection results using different dividers.

B. Foreground extraction

Foreground extraction plays a key role in computer vision. It is used to remove the background from an image to obtain the foreground object clearly. However, it is not efficient to use the exact divider in foreground extraction as the computation time increases tremendously when processing high-resolution images. The approximate divider becomes an alternative to solve this problem, which can acquire a decent result with very limit time. Similar to change detection, a division between two pixels of the same position in two images is performed, however a higher accuracy is required. Thus, in this experiment, the half-precision (16-bit) dividers are used. The ALD (10), FaNZeD (10), TrunApp4 (4), FPAD33 (10), FPAD43 (10), AAXD (10), LPCAD (2, 10) and LPCAD (3, 10) are considered in this application.

Three images are evaluated to acquire the average PSNRs and SSIMs for all approximate designs. The results are listed in Table VI. It shows that LPCAD (3, 10) and LPCAD (2, 10) outperform the other approximate dividers in terms of PSNR. The SSIMs also show the superiority of the proposed dividers over state-of-the-art designs. We further explore the impact of the parameters k and t for the LPCAD; the PSNR and SSIM are shown in Table VII. It can be seen that the output image is more explicit with a higher t for the LPCAD. The quality of the output images reaches to the highest with $t = 6$ and then remains stable. Thus, the LPCAD (3, 6) is the best trade-off for foreground extraction.

C. K-Means color quantization

Color quantization is performed to reduce the number of distinct colors used in an image and obtain a compressed image as similar as to the original one. Color quantization enables the transmission of the images in devices supporting only a limited number of color types, such as a storage with

TABLE VI: The PSNRs (dB) / SSIMs of the foreground extraction results using different approximate dividers.

Designs	son	canoe	office	average
LPCAD (3, 10)	38 / 0.96	50 / 0.99	48 / 0.99	45 / 0.98
LPCAD (2, 10)	35 / 0.95	46 / 0.99	43 / 0.98	41 / 0.97
ALD (10)	32 / 0.95	38 / 0.98	36 / 0.97	35 / 0.96
FaNZeD (10)	29 / 0.94	41 / 0.98	41 / 0.97	37 / 0.96
TrunApp4 (4)	23 / 0.78	37 / 0.96	37 / 0.92	32 / 0.88
FPAD33 (10)	21 / 0.84	39 / 0.98	39 / 0.95	33 / 0.92
FPAD43 (10)	26 / 0.86	42 / 0.98	42 / 0.96	36 / 0.93
AAXD (10)	31 / 0.88	45 / 0.99	45 / 0.98	40 / 0.95

TABLE VII: The PSNRs/SSIMs of the foreground extraction results using LPCAD for $k = 3$ with different t values.

t	2	4	6	8	10
son	16 / 0.57	30 / 0.86	38 / 0.96	38 / 0.96	38 / 0.96
canoe	29 / 0.88	41 / 0.96	49 / 0.99	50 / 0.99	50 / 0.99
office	29 / 0.86	40 / 0.98	47 / 0.99	48 / 0.99	48 / 0.99
average	25 / 0.77	37 / 0.93	45 / 0.98	45 / 0.98	45 / 0.98

limited capacity. The key problem for quantization is to find the proper color palette that summarizes the original image best. A common approach for color quantization is based on the K-Means algorithm. This algorithm divides the given pixel values of the image into K clusters according to their distances from the cluster center that is computed recurrently. In each iteration, the center of each cluster is updated by dividing the number of pixel values of this cluster by their summation. Due to the iterations, a relatively high accuracy is required in K-Means compared to the former two applications. Thus, single-precision (32-bit) FP dividers are utilized to complete this task. The ALD (23), FaNZeD (23), TrunApp4 (4), FPAD33 (23), FPAD43 (23), AAXD (10), LPCAD (2, 8) and LPCAD (3, 8) are considered.

The simulation results are shown in Table VIII, where three images are examined. The LPCAD (3, 8) and LPCAD (2, 8) perform significantly better than the other dividers in terms of PSNR, followed by FPAD43 (23), AAXD (10), FaNZeD (23), TrunApp4 (4), FPAD33 (23) and ALD (23). As per SSIMs, the LPCAD (3, 8) significantly outperforms the other dividers. It should be noted that the image 'kodim' is quantized with ten different colors and needs more accurate approximate dividers. The LPCAD (3, 8) produces the highest PSNR and SSIM for image 'kodim'.

VII. Conclusion

In this paper, an energy-efficient and high-performance approximate division scheme based on logarithmic conversion and piecewise constant approximation is proposed for error-resilient applications. To minimize the statistical error, a heuristic search algorithm is designed to obtain the most appropriate constant set to approximate the reciprocal of the divisor. The hardware implementations for both FP and integer divisions are presented. The simulation and synthesis results show that the proposed FP divider outperforms state-of-the-art approximate dividers when considering both the accuracy and hardware. The obtained configurations of the proposed divider are in the Pareto Frontier in terms of PDP and MRED. Also, the proposed divider results in the highest processing quality in three image processing applications. Finally, it is worth to note that the quality of the proposed divider is configurable and a trade-off can be made between the accuracy and hardware overhead according to the requirement of specific applications.

TABLE VIII: The PSNRs and SSIMs of the color quantization results using different approximate dividers.

Designs	mandm	peppers	kodim	average
LPCAD (3, 8)	37 / 0.99	33 / 0.99	34 / 0.93	34 / 0.97
LPCAD (2, 8)	37 / 0.99	32 / 0.99	27 / 0.77	32 / 0.92
ALD (23)	24 / 0.96	26 / 0.97	26 / 0.77	25 / 0.90
FaNZeD (23)	28 / 0.98	27 / 0.97	27 / 0.80	27 / 0.92
TrunApp4 (4)	25 / 0.97	28 / 0.98	27 / 0.80	26 / 0.91
FPAD33(23)	21 / 0.91	29 / 0.98	24 / 0.71	25 / 0.87
FPAD43(23)	23 / 0.96	33 / 0.99	27 / 0.84	28 / 0.93
AAXD(10)	24 / 0.97	31 / 0.99	26 / 0.78	27 / 0.91

References

[1] V. Leon, K. Asimakopoulos, S. Xydis, D. Soudris, and K. Pekmestzi, "Cooperative arithmetic-aware approximation techniques for energy-efficient multipliers," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[2] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2108–2135, 2020.

[3] B. Liu, X. Ding, H. Cai, W. Zhu, Z. Wang, W. Liu, and J. Yang, "Precision adaptive mfcc based on r2sdf-fft and approximate computing for low-power speech keywords recognition," *IEEE Circuits and Systems Magazine*, vol. 21, no. 4, pp. 24–39, 2021.

[4] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," *SIGPLAN Not.*, vol. 47, no. 4, pp. 301–312, 2012.

[5] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–9.

[6] S. Hashemi, N. Anthony, H. Tann, R. I. Bahar, and S. Reda, "Understanding the impact of precision quantization on the accuracy and energy of neural networks," in *2017 54th Design, Automation Test in Europe Conference Exhibition (DATE)*, 2017, pp. 1474–1479.

[7] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design(ISLPED)*, 2016, pp. 64–69.

[8] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," in *Micro*, 2013, pp. 1–12.

[9] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS)*, 2013, pp. 1–6.

[10] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[11] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, 2016.

[12] M. Gautschi, M. Schaffner, F. K. Gürkaynak, and L. Benini, "An extended shared logarithmic unit for nonlinear function kernel acceleration in a 65-nm cmos multicore cluster," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 98–112, 2017.

[13] B. Liu, H. Cai, Z. Wang, Y. Sun, Z. Shen, W. Zhu, Y. Li, Y. Gong, W. Ge, J. Yang, and L. Shi, "A 22nm, 10.8 μw/15.1 μw dual computing modes high power-performance-area efficiency domained background noise aware keyword-spotting processor," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 4733–4746, 2020.

[14] A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: A new paradigm for arithmetic circuit design," in *2008 Design, Automation and Test in Europe*, 2008, pp. 1250–1255.

[15] F. Ebrahimi-Azandaryani, O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Block-based carry speculative approximate adder for energy-efficient applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 1, pp. 137–141, 2020.

[16] D. Esposito, A. G. M. Strollo, E. Napoli, D. De Caro, and N. Petra, "Approximate multipliers based on new approximate compressors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 12, pp. 4169–4182, 2018.

[17] R. Pilipović, P. Bulić, and U. Lotrič, "A two-stage operand trimming approximate logarithmic multiplier," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 6, pp. 2535–2545, 2021.

[18] L. Chen, J. Han, W. Liu, and F. Lombardi, "Design of approximate unsigned integer non-restoring divider for inexact computing," in *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI(GLSVLSI)*, 2015, pp. 51–56.

[19] ——, "On the design of approximate restoring dividers for error-tolerant applications," *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2522–2533, 2016.

[20] W. Liu, T. Xu, J. Li, C. Wang, P. Montuschi, and F. Lombardi, "Design of unsigned approximate hybrid dividers based on restoring array and logarithmic dividers," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 1, pp. 339–350, 2022.

[21] L. Chen, J. Han, W. Liu, P. Montuschi, and F. Lombardi, "Design, evaluation and application of approximate high-radix dividers," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 3, pp. 299–312, 2018.

[22] S. Hashemi, R. I. Bahar, and S. Reda, "A low-power dynamic divider for approximate applications," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.

[23] H. Jiang, L. liu, F. Lombardi, and J. Han, "Low-power unsigned divider and square root circuit designs using adaptive approximation," *IEEE Transactions on Computers*, vol. 68, no. 11, pp. 1635–1646, 2019.

[24] J. Melchert, S. Behroozi, J. Li, and Y. Kim, "SAADI-EC: A quality-configurable approximate divider for energy efficiency," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2680–2692, 2019.

[25] S. Vahdat, M. Kamal, A. Afzali-Kusha, M. Pedram, and Z. Navabi, "Truncapp: A truncation-based approximate divider for energy efficient DSP applications," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 1635–1638.

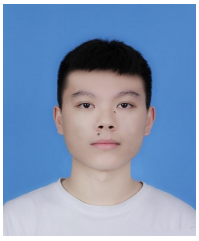
[26] C. K. Jha, K. Prasad, V. K. Srivastava, and J. Mekie, "FPAD: A multistage approximation methodology for designing floating point approximate dividers," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[27] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, vol. EC-11, no. 4, pp. 512–517, 1962.

[28] H. Saadat, H. Javaid, and S. Parameswaran, "Approximate integer and floating-point dividers with near-zero error bias," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[29] M. Schulte and E. Swartzlander, "Truncated multiplication with correction constant [for dsp]," in *Proceedings of IEEE Workshop on VLSI Signal Processing*, 1993, pp. 388–396.

[30] R. Radke, S. Andra, O. Al-Kofahi, and B. Roysam, "Image change detection algorithms: a systematic survey," *IEEE Transactions on Image Processing*, vol. 14, no. 3, pp. 294–307, 2005.



Yong Wu received the B.S. degree in electronical information science and technology from the University of Electronic Science and Technology of China, Chengdu, China in 2020. He is currently pursuing the master's degree at the School of Integrated Circuits, Tsinghua University, Beijing, China. His current research interests include approximate computing and reconfigurable computing.

Honglan Jiang (S'14-M'18) received the B.Sc. and master's degrees in instrument science and technology from the Harbin Institute of Technology, Harbin, China, in 2011 and 2013, respectively, and the Ph.D. degree in integrated circuits and systems from the University of Alberta, Edmonton, AB, Canada, in 2018. From 2018 to 2021, she was a Postdoctoral Fellow with the School of Integrated Circuits, Tsinghua University, Beijing, China. She is currently an associate professor with the Department of Micro-Nano Electronic, Shanghai Jiao Tong University, Shanghai, China. Her research interests include approximate computing, reconfigurable computing, and stochastic computing.

Zining Ma received the B.S. degree in microelectronics from the Harbin Institute of Technology, Harbin, China in 2019. He is currently studying forward the master's degree at the School of Integrated Circuits, Tsinghua University, Beijing, China. His current research interests include stochastic computing and reconfigurable computing.

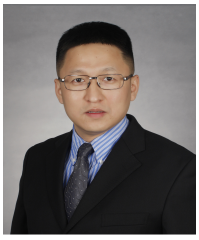
Pengfei Gou received the B.S., M.S. and Ph.D. degrees in Microelectronics from Harbin Institute of Technology, Harbin, China, in 2006, 2008 and 2012 respectively. He is currently the senior architect of Hexin Technology. His research interests include high performance processor architecture, heterogeneous computing architecture and VLSI SoC Design.

Yong Lu received the B.S. and M.S. degrees in Electronics Engineering from Shanghai Jiao Tong University, Shanghai, China, in 2004 and 2007 respectively. She is a senior architect of Hexin Technology and focusing on the chip architecture of high-performance processors, especially for the reliability, security, visibility, power and performance management. Her research interests also include heterogeneous computing and VLSI verification methodology.

Jie Han (S'02-M'05-SM'16) received the B.Sc. degree in electronic engineering from Tsinghua University, Beijing, China, in 1999 and the Ph.D. degree from the Delft University of Technology, The Netherlands, in 2004. He is currently a Professor in the Department of Electrical and Computer Engineering at the University of Alberta, Edmonton, AB, Canada. His research interests include approximate computing, stochastic computing, reliability and fault tolerance, nanoelectronic circuits and systems, novel computational models for nanoscale and biological applications. Dr. Han was a recipient of the Best Paper Award at the International Symposium on Nanoscale Architectures (NanoArch) 2015 and Best Paper Nominations at the 25th Great Lakes Symposium on VLSI (GLSVLSI) 2015, NanoArch 2016 and the 19th International Symposium on Quality Electronic Design (ISQED) 2018. He was nominated for the 2006 Christiaan Huygens Prize of Science by the Royal Dutch Academy of Science. His work was recognized by *Science*, for developing a theory of fault-tolerant nanocircuits (2005). He is currently an Associate Editor for the IEEE Transactions on Emerging Topics in Computing (TETC), the IEEE Transactions on Nanotechnology, the IEEE Circuits and Systems Magazine, the IEEE Open Journal of the Computer Society and Microelectronics Reliability (Elsevier Journal). He served as a General Chair for GLSVLSI 2017 and the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT) 2013, and a Technical Program Committee Chair for GLSVLSI 2016, DFT 2012 and the Symposium on Stochastic & Approximate Computing for Signal Processing and Machine Learning, 2017.

Shouyi Yin (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electronic engineering from Tsinghua University, China, in 2000, 2002, and 2005, respectively.,He worked at Imperial College London, London, U.K., as a Research Associate. He is currently a Processor with the School of Integrated Circuits, Tsinghua University. His research interests include reconfigurable computing, mobile computing, and system-on-chip (SoC) design.

Shaojun Wei was born in Beijing, China in 1958. He received Ph.D. degree from Faculte Polytechnique de Mons, Belguim, in 1991. He became a professor in Institute of Microelectronics of Tsinghua University in 1995. He is a senior member of Chinese Institute of Electronics (CIE). His main research interests include VLSI SoC design, EDA methodology, and communication ASIC design.



Leibo Liu (M'10-SM'17) received the B.S. degree in electronic engineering and the Ph.D. degree with the Institute of Microelectronics, both from Tsinghua University, Beijing, China, in 1999 and 2004, respectively. He is currently a Full Professor with the School of Integrated Circuits, Tsinghua University. His current research interests include reconfigurable computing, mobile computing, and very large-scale integration digital signal processing.