PACE: A Piece-Wise Approximate and Configurable Floating-Point Divider for Energy-Efficient Computing

Chenyi Wen[1], Haonan Du[1], Zhengrui Chen[1], Li Zhang[2,*], Qi Sun[1], Cheng Zhuo[1,3,*]
[1]*Zhejiang University*
[2]*Hubei University Of Technology*
[3]*Key Laboratory of Collaborative Sensing and Autonomous Unmanned Systems of Zhejiang Province, China*
[*]Corresponding author: zhangli@hbut.edu.cn, czhuo@zju.edu.cn

***Abstract*—Approximate computing is a promising alternative to improve energy efficiency for human perception related applications on the edge. This work proposes a piece-wise approximate floating-point divider, which is resource-efficient and run-time configurable. We provide a piece-wise approximation algorithm for $1/y$, utilizing powers of 2. This approach enables the implementation of a reciprocal-based floating-point divider that is independent of multipliers, which not only reduces hardware consumption but also results in shorter latency. Furthermore, a multi-level run-time configurable hardware structure is introduced, enhancing the adaptability to various application scenarios. When compared to the prior state-of-the-art approximate divider, the proposed divider strikes an advantageous balance between accuracy and resource efficiency. The application-level evaluation of the proposed dividers demonstrates manageable and minimal degradation of the output quality when compared to the exact divider.**

***Index Terms*—Approximate computation, Divider, Floating-Point, Energy Efficiency**

I. Introduction

Recently, with the growing popularity of human perception related tasks ($e.g.$, audio/image processing, machine learning), it is observed that the requirements of full precision and exactness are not always essential [1]–[3]. This observation has led to growing interest in approximate computing, a method that trades off computational accuracy for energy efficiency [4], [5]. Thus, approximate arithmetic units like adders and multipliers have been explored to reduce power consumption with a tolerable loss in accuracy [6], [7].

Among the four basic arithmetic operations, division is generally more time-consuming and energy-intensive [8]. Despite this, division is crucial in various applications, such as the OpenCV library in multimedia [9], normalization in page-ranking [10], and even derivatives in robotics control [11], etc. **There are significant demands to resource-efficient dividers** for applications, where division cannot be bypassed.

Recent works have proposed several types of approximate dividers, which can be grouped into three main categories: subtractor-array-based [12]–[15], logarithm-based [16]–[20], and reciprocal-based dividers [8], [21]–[25]. However, each category of the existing designs has its own limitations. Subtractor-array-based and reciprocal-based dividers can achieve high computational accuracy, but often **impose substantial hardware overhead**. Subtractor-array-based dividers face high latency issues due to their array structure [12]–[15], while reciprocal-based designs often require multiple (1-3) multipliers for completing computations [21]–[25]. Nevertheless, since an exact floating-point (FP) divider incurs more than $15\times$ Area-Delay-Product and over $1.5\times$ power consumption compared to an exact FP multiplier [17], the above approximate dividers are still beneficial in hardware resource efficiency. On the other hand, logarithm-based designs are more hardware-efficient but **struggle with accuracy and configurability during runtime** [16]–[20]. Usually, logarithm-based designs rely on pre-loaded lookup tables (LUTs), making runtime adjustments to accuracy impossible [19], [20].

To address the above issues, we propose PACE (a **P**iece-wise **A**pproximate and **C**onfigurabl**E** FP divider), a resource-efficient and run-time configurable design. Our major contributions include:

- We implemented **a multiplier-independent reciprocal-based floating-point divider**, which replaced the multiplication by $1/y$ with much simpler shifts and additions, reducing both hardware resource demands and latency.
- **A piece-wise approximation algorithm** is introduced for $1/y$ operation. By increasing the number of piece-wise linear segments to approximate $1/y$, this method easily improves overall division accuracy.
- Finally, **a multi-level run-time configurable circuit structure** is designed for greater adaptability across various applications. Based on the application specifications, designers are empowered to decide approximation levels at run-time to balance delay and energy consumption.

The experimental results clearly demonstrate that, in comparison to state-of-the-art dividers, PACE strikes an advantageous balance between accuracy and resource efficiency. Specifically, PACE achieves comparable accuracy to the most precise approximate divider, QIAD [25], while consuming a mere 11.61% of the resources in Area-Delay-Product. FaNZeD [17] is the only other divider that occupies less hardware resource than PACE, but its errors are about $5\times$ greater than PACE's. The application-level evaluation of PACE demonstrates marginal degradation of the output quality, resulting in a mere 0.09dB average PSNR reduction in K-means color quantization and a 0.63dB average PSNR reduction in JPEG.

979-8-3503-4859-0/DATE24/© 2024 EDAA

II. Related Work

Early approaches to approximate dividers often simply modified the conventional, precise array dividers. The work in [12], [15] replaced core units in non-restoring and high-radix dividers with approximate units. The approximation level could be tuned by adjusting the extent of the replacement. Another strategy, as seen in [13], [14], added a pruning circuit to a standard divider. This circuit truncated the input to fewer bits after identifying the leading one, thereby simplifying the existing divider module. However, these approaches are limited by the underlying principles of precise division, leading to hardware redundancy and sub-optimal approximation performance.

Then the approximation techniques leveraging the principle of logarithms have been proposed, known as the Mitchell Method [16], in which complex division can be converted to the straightforward subtraction in the logarithmic domain. A typical logarithmic divider consists of three main stages: logarithmic conversion, subtraction, and anti-logarithmic conversion. To facilitate binary operations, logarithmic calculations are approximated as linear functions, as outlined in Eq. (1):

$$\log N = k + \log(1+x) \approx k + x, \tag{1}$$

where $N = 2^k(1+x), 0 \leqslant x < 1$. Despite its simplicity, the logarithmic divider introduces non-negligible error. To offset this, FaNZeD [17] and LEAD [18] used compensation coefficients and exponential series expansions, respectively, to improve the accuracy.

An alternative category of dividers transforms division A/B into multiplication $A \times (1/B)$, as multiplication is generally more resource-efficient [25]. Various methods have been proposed to compute the reciprocal $1/B$ [25]. For example, ILAFD [21] directly applied bit-wise inversion, while TruncApp [8] used a specific approximation for $1/X_B \approx \overline{((X_B+1)/2)}$, with $X_B \in [1,2)$ denoting the normalized fractional component of B. SEERAD [22] pre-computed the optimal parameters for transforming the reciprocal into multiplication and shift operations. While these dividers are simple, they often incur higher error rates and typically require lookup tables to store parameters or error-compensation values.

Finally, some researches have focused on the mathematical approximations of the function $1/x$. For example, SAADI-EC [23] employed Taylor expansions, ADNR-EC [24] used Newton's iteration, and QIAD [25] utilized quadratic curve fitting. These methods offer high precision but the mathematical approximations typically require multiple multipliers within one clock cycle or additional clock cycles using one multiplier.

In summary, while there has been considerable progress in the field, **finding an approximate divider that balances both precision and resource efficiency remains a challenge**.

III. Proposed Approximate Divider Design

A. Proposed Approximate Division

According to the IEEE 754 standard for FP arithmetic, any FP number F_x can be represented as below:

$$F_x = sign_x \times x \times 2^{E_x} \tag{2}$$

where $sign_x$, x, and E_x refer to the sign, mantissa, and exponent of the FP number F_x, with $x \in [1,2)$. For the division between two FP numbers F_x and F_y, we have:

$$F_x/F_y = sign_x sign_y \times (x/y) \times 2^{E_x - E_y} \tag{3}$$

Given that computing the mantissa x/y is much more energy- and delay-consuming than the other two parts, its energy efficient design is the focus of the rest of the paper.

To alleviate the computational complexity of x/y, we propose a transformation as presented in Equation (4):

$$x/y = (1/y) \times (x-y) + 1 \tag{4}$$

Here, we approximate $1/y$ based on the value of y, and as a result, Eq. (4) intrinsically leverages a linear plane to interpolate the surface of x/y. Now assume that y falls within the interval $[y_1, y_2)$, with y_1 and y_2 defined as:

$$y_1 = 1 + f_1 = 1 + \frac{t}{2^n} \tag{5}$$

$$y_2 = 1 + f_2 = 1 + \frac{t+1}{2^n} \tag{6}$$

where $f_1, f_2 \in [0,1)$, n refers to approximate level, and t depends on y.

The mean value of $1/y$ across the range of $[y_1, y_2]$ can be denoted as y_n^t:

$$y_n^t = \frac{1}{y_2 - y_1} \int_{y_1}^{y_2} \frac{1}{y} dy = \frac{\ln y_2 - \ln y_1}{y_2 - y_1} \tag{7}$$

Then, to further simplify Eq. (7), we employ the Taylor series expansion of $\ln(1+f)$ shown in Eq. (8):

$$\ln(1+f) = 1 + f - \frac{f^2}{2} + \frac{f^3}{3} + ... + (-1)^{k+1}\frac{f^k}{k} \tag{8}$$

We substitute Eq. (8) into Eq. (7):

$$\begin{aligned} y_n^t &= \frac{\ln(1+f_2) - \ln(1+f_1)}{(1+f_2) - (1+f_1)} \\ &= \frac{1}{f_2 - f_1} \times (f_2 - f_1 - \frac{f_2^2 - f_1^2}{2} + \frac{f_2^3 - f_1^3}{3} - ...) \\ &= 1 - \frac{f_2 + f_1}{2} + \frac{f_2^2 + f_1^2 + f_1 f_2}{3} - ... \end{aligned} \tag{9}$$

The first two terms of Eq. (9), namely 1 and $-\frac{f_2+f_1}{2}$, are retained in their original form. For computational simplicity, the remaining terms are approximated as $\frac{f_1 f_2}{2}$. This approximation is chosen because it captures the first-order behavior of the remaining terms while accounting for hardware friendliness. The approximated expression for y_n^t is denoted as k_n^t:

$$y_n^t \approx k_n^t = 1 - \frac{f_2 + f_1}{2} + \frac{f_1 f_2}{2} = 1 - \frac{2t+1}{2^{n+1}} + \frac{t^2+t}{2^{2n+1}} \tag{10}$$

Here, n and t have the same meanings as in Eq.(5) and Eq.(6). By substituting this approximation into Eq. (4), we arrive at

$$x/y \approx y_n^t \times (x-y) + 1 \approx k_n^t \times (x-y) + 1 \tag{11}$$

To aid in further analysis, let f_{approx}^n represent the approximate mantissa division at the approximation level n:

$$f_{approx}^n = k_n^t \times (x-y) + 1 \tag{12}$$

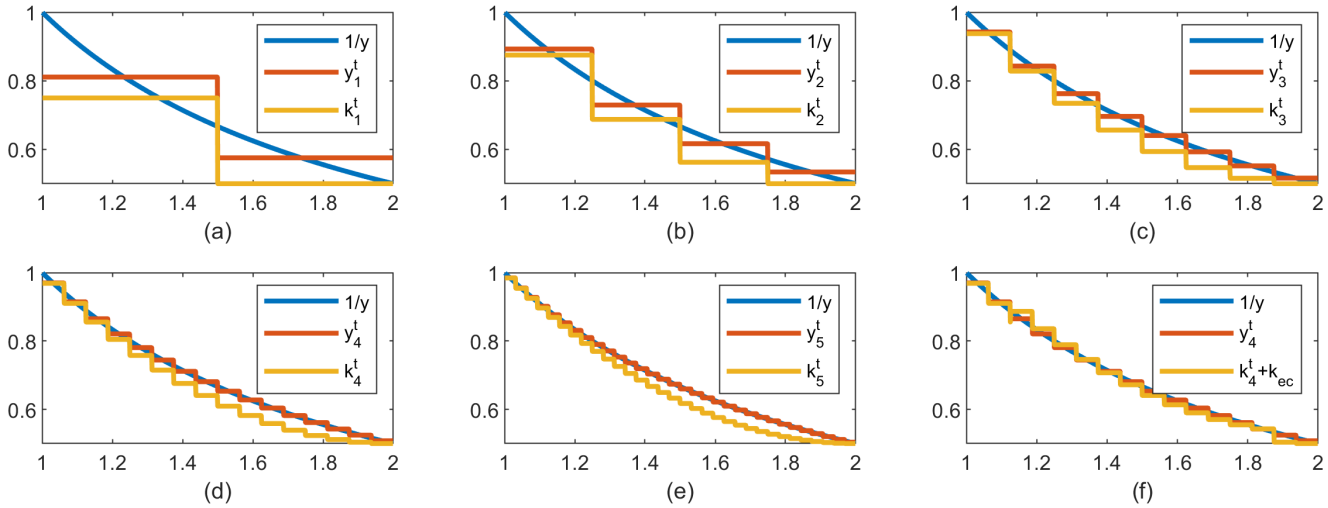

Fig. 1. Values of $1/y$, y_n^t, and k_n^t for different approximate levels n, where y is in the range of [1,2): (a) approximate level $n = 1$; (b) approximate level $n = 2$; (c) approximate level $n = 3$; (d) approximate level $n = 4$; (e) approximate level $n = 5$; (f) approximate level $n = 4$ with error compensation k_{ec}.

With the above mathematical formulations, we lay the groundwork for an energy-efficient and computationally simpler method to approximate division.

B. Error Compensation

The accuracy of the approximate division is characterized by the difference between y_n^t and k_n^t. Fig. 1(a)-(e) compare the values of $1/y$, y_n^t, and k_n^t, as the approximate level n varies from 1 to 5. To quantitatively assess the disparity between y_n^t and k_n^t, we define the mean error y_n^{err} between them at each level n as follows:

$$y_n^{err} = \frac{1}{2^n}\sum_{t=0}^{n-1}(y_n^t - k_n^t) \tag{13}$$

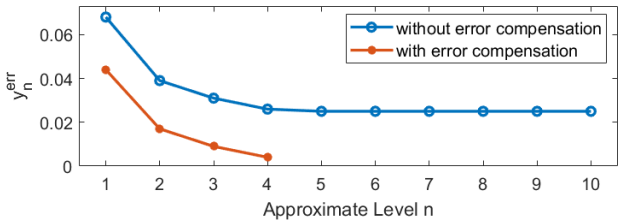

Fig. 2. Values of y_n^{err} (with/without error compensation) when approximate level n varies.

Fig. 2 reveals that y_n^{err} significantly decreases as n increases from 1 to 4. Beyond $n = 4$, however, the error plateaus, suggesting that further levels of approximation are not beneficial for accuracy. This motivates **the need for error compensation mechanisms at the fourth or lower level of approximation.**

For example, Fig. 1(d) shows that k_4^t closely approximates y when y is near 1 or 2 but is generally underestimated otherwise. An effective solution for error compensation involves adding a constant term $k_{ec,4}$ to k_4^t:

$$k_{ec,4} = \begin{cases} 0, & y < 1.125 \text{ or } y \geq 1.875 \\ \dfrac{1}{2^5}, & 1.125 \leq y < 1.875 \end{cases} \tag{14}$$

Fig. 1(f) and 2 show that introducing k_{ec} significantly reduces y_4^{err} compared to y_5^{err} without compensation.

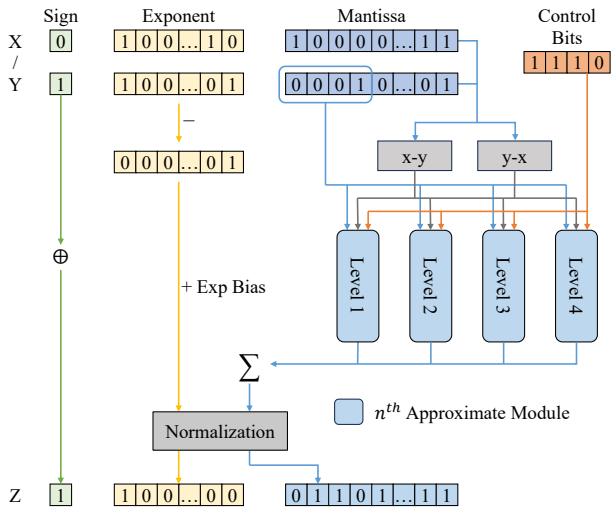

Fig. 3. Architecture of the proposed approximate divider.

IV. Circuit Implementation

A. Multi-Level Approximation Architecture

In the preceding sections, we introduced a novel method for approximate division that enhances computational accuracy through piece-wise approximation. This section further presents a multi-level architecture designed to implement this algorithm, allowing for dynamic configuration at run-time, which is illustrated in Fig. 3.

Inside the module designated for mantissa division, two initial subtractors calculate $x-y$ and $y-x$. Their outputs, along with the mantissa y and control bits, feed into a multi-level computation structure. In this structure, Level 1 serves as the fundamental approximation unit, producing an initial estimate f_{approx}^1. Subsequent levels act as deviation correction layers, iteratively refining the approximation and thereby enhancing overall accuracy. This architecture enables simple run-time configuration by selecting the desired number of layers.

To facilitate the following discussions, we define f_{ec}^n as the discrepancy between consecutive approximation levels, n and

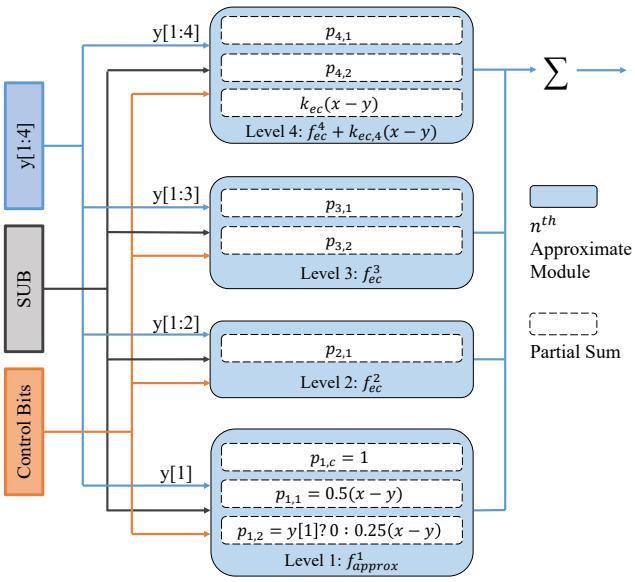

Fig. 4. Detailed Architecture of the multi-level mantissa computation.

$n-1$. Thus, f_{ec}^n represents the output from the n^{th} level. The proposed architecture in Fig. 3 simply implements an accuracy-configurable approximation through the control of approximation granularity as:

$$f_{approx}^n = f_{approx}^1 + \sum_{i=2}^{n} f_{ec}^i \tag{15}$$

We can easily compute f_{approx}^1 by Eq. (10) and (12):

$$f_{approx}^1 = \begin{cases} 1 + \frac{1}{2}(x-y) + \frac{1}{4}(x-y), & y \le 1.5 \\ 1 + \frac{1}{2}(x-y), & y > 1.5 \end{cases} \tag{16}$$

Then we can calculate f_{ec}^n based on Eq. (12):

$$f_{ec}^n = (k_n^t - k_{n-1}^{floor(t/2)}) \times (x-y) \tag{17}$$

where $floor(.)$ is the round-down function. For the convenience of following discussion, we denote the deviation correction of k_n^t between different levels as K_n^t:

$$K_n^t = k_n^t - k_{n-1}^{floor(t/2)} = \begin{cases} \frac{1}{2^{n+1}} - \frac{t}{2^{2n+1}}, & t \ mod\ 2 = 0 \\ -\frac{1}{2^{n+1}} + \frac{t+1}{2^{2n+1}}, & t \ mod\ 2 = 1 \end{cases} \tag{18}$$

The values of all K_n^t's for $n = 2, 3, 4$ can be readily derived depending on the first 4 digits of y's mantissa, as shown in Table I. The architecture capitalizes on the fact that all K_n^t values consist solely of powers of 2. This simplifies the calculation of f_{ec}^n to basic shift and add operations. The architecture uses the notion of Partial Sums (**PS**) for level-wise configuration. Depending on the value of K_n^t, Level 2 requires a single PS configuration, followed by two for Level 3 and two for Level 4. Level 1 requires three PS configurations, and an additional PS is needed for the term k_{ec}. The elaborate

TABLE I
RELATIONSHIP BETWEEN THE FIRST 4 DIGITS OF y MANTISSA AND K_n^t.

y[1:4]*	K_n^t		
	$n=2$	$n=3$	$n=4$
0000	$K_2^0 = 2^{-3}$	$K_3^0 = 2^{-4}$	$K_4^0 = 2^{-5}$
0001			$K_4^1 = -2^{-5} + 2^{-8}$
0010		$K_3^1 = -2^{-4} + 2^{-6}$	$K_4^2 = 2^{-5} - 2^{-8}$
0011			$K_4^3 = -2^{-5} + 2^{-7}$
0100	$K_2^1 = -2^{-4}$	$K_3^2 = 2^{-4} - 2^{-6}$	$K_4^4 = 2^{-5} - 2^{-7}$
0101			$K_4^5 = -2^{-6} - 2^{-8}$
0110		$K_3^3 = -2^{-5}$	$K_4^6 = 2^{-6} + 2^{-8}$
0111			$K_4^7 = -2^{-6}$
1000	$K_2^2 = 2^{-4}$	$K_3^4 = 2^{-5}$	$K_4^8 = 2^{-6}$
1001			$K_4^9 = -2^{-7} - 2^{-8}$
1010		$K_3^5 = -2^{-6}$	$K_4^{10} = 2^{-7} + 2^{-8}$
1011			$K_4^{11} = -2^{-7}$
1100	$K_2^3 = 0$	$K_3^6 = 2^{-6}$	$K_4^{12} = 2^{-7}$
1101			$K_4^{13} = -2^{-8}$
1110		$K_3^7 = 0$	$K_4^{14} = 2^{-8}$
1111			$K_4^{15} = 0$

*The first 4 digits of y mantissa.

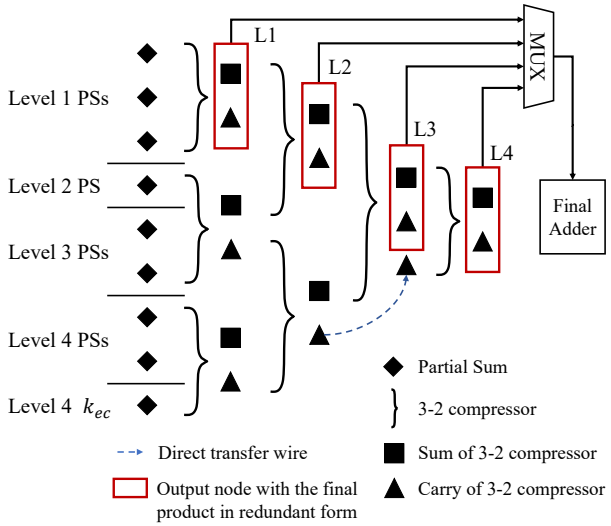

Fig. 5. Partial sum reduction tree structure for the proposed divider with run-time configurability.

circuitry for mantissa computation is depicted in Fig. 4, where $p_{n,i}$ is employed to represent the ith partial sum production at the level n.

B. Design for Run-Time Configurability

To enhance the efficiency of partial sum accumulations, we propose a novel tree structure that employs 3-to-2 compressors. This innovative design allows for dynamic switching between various levels of approximation within a single circuit. Fig. 5 details this structure. In this figure, each dot signifies a partial sum, brackets denote 3-to-2 compressors, and squares and triangles indicate the sum and carry outputs from these compressors, respectively. A dotted arrow represents a direct wire that transfers a partial sum to a subsequent stage.

Notably, the tree features four output nodes, highlighted in red rectangles. These correspond to approximation levels 1, 2, 3, and a 4_{th} level that includes an error compensation

term, k_{ec}, which is determined based on the chosen approximation level. When operating at lower approximation levels, the architecture bypasses the need to navigate through the entire tree. This not only simplifies computational tasks but also minimizes delays along the critical path. Such design versatility renders the architecture highly amenable to run-time configurability. It's important to emphasize that the inclusion of the k_{ec} term in the Level 4 approximation module does not extend the circuit's critical path. Thus, we strongly recommend employing this error compensation term to improve accuracy. For the sake of clarity, all the further discussions in this paper will use "L4" to signify Level 4 approximation, denoted as f^4_{approx}, which includes the $k_{ec,4}$ term.

V. Evaluation and Discussion

We evaluated the proposed divider across four approximation levels (L1-L4), focusing on accuracy, resource utilization, and performance. For comparison, we also tested several state-of-the-art approximate dividers, including TruncApp [8], FaNZeD [17], LEAD [18], and QIAD [25]. Some dividers were proposed as fixed point dividers but can be easily converted to FP dividers by removing the LOD modules and the barrel shifters.

A. Accuracy Comparison

To evaluate the accuracy of the proposed divider, we uniformly sampled 10,000 points within the range of [1, 2) to serve as both divisors x and dividends y. We then tested each possible combination of x and y, and compared the results with those from an exact FP divider. We employed three error metrics for evaluation: Mean Absolute Error (MAE), Mean Relative Error Distance (MRED), and Root Mean Square Error (RMSE). The results are shown in the first three columns of Table II. Clearly, as the approximation level increases, the accuracy of the PACE improves. For most metrics, PACE outperforms FaNZeD [17] and TruncApp [8] even with L1 approximation. PACE is superior to LEAD from level 2. While QIAD [25] achieves high accuracy, PACE at L4 can achieve similar accuracy with much smaller resource cost.

B. Resource Utilization and Performance Comparison

To assess hardware resource utilization and performance, we implemented all the dividers in 32-bit FP using Verilog HDL. All the designs were synthesized and evaluated using the Synopsys Design Compiler with UMC's 40-nm library, with consistent design constraints and options. The exact 32-bit FP divider and multiplier IPs from Synopsys DesignWare library were employed for comparison purpose.

Columns 6-9 in Table II display the hardware resource and performance results, including area, delay, power, and normalized area-delay-product (Norm. ADP). We use the exact 32-bit FP divider from the Synopsys DesignWare library as a reference, normalizing its results to 1. Even at its most resource-intensive level (L4), PACE is 70% smaller and 65% faster than the exact divider. In fact, PACE(L4) outperforms even the exact multiplier IP in size and speed.

Compared to other approximate dividers, PACE also demonstrates significant advantages in area, delay, and power. For example, PACE(L4) uses only a third of the area and is three times faster than QIAD [25], which offers similar accuracy. FaNZeD [17] is the only other divider that takes up less area than PACE(L4), but its errors are about 5-6 times greater. Among prior works, LEAD [18] is the most balanced in terms of area, delay, power, and accuracy. However, compared to PACE(L4), LEAD is 70% larger, 120% slower, and incurs nearly 4× larger errors.

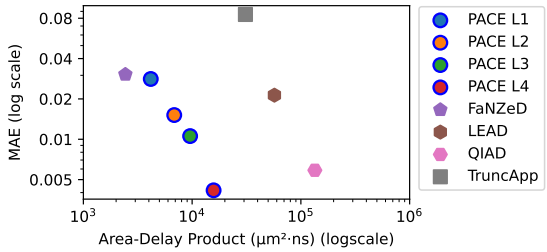

Fig. 6. Comparison on ADP v.s. MAE among PACE and prior works.

Fig. 6 compared ADP and MAE to further illustrate PACE's superior performance. Data points closer to the origin indicate a better trade-off between accuracy and resource utilization. PACE achieves substantial accuracy gains with minimal resource overhead. Compared to LEAD [18], PACE's L4 implementation shows a 72.41% improvement in ADP and an 76.19% increase in MAE-based accuracy. Moreover, PACE matches the near-lossless accuracy of QIAD [25] while using only 11.61% of the ADP resources.

C. Application-Level Comparison

To assess the performance of approximate dividers in real-world applications, similar as prior works [17], [18], [21], [23]–[25], we integrated them into two multimedia application algorithms that heavily depend on division operations: K-means color quantization and JPEG. In K-means color quantization, we employed the approximate dividers for the standard cluster averaging operation in each K-means iteration [18]. In JPEG, the element-wise division in DCT was replaced with the approximate dividers to quantize the DCT coefficients [17].

We evaluated the modified algorithms on the Caltech101 dataset [26], which contains 102 categories and 9144 images in total. To measure the output quality, we calculated the difference in Peak Signal-to-Noise Ratio (PSNR) between images processed with approximate and accurate dividers, denoted as $\text{PSNR}_{\text{degradation}} = \text{PSNR}_{\text{accurate}} - \text{PSNR}_{\text{approx}}$.

Fig. 7 shows the PSNR degradation due to the use of approximate dividers on the Caltech101 dataset. As the approximation level of PACE increases, image quality consistently improves. At level 2, PACE outperforms most other approximate dividers for both applications, except for QIAD [25]. At level 4, PACE exhibits a mere 0.09dB average reduction in K-means and a 0.63dB average reduction in JPEG compared to the exact divider. Notably, PACE(L4) achieves better quality in both two applications when compared to QIAD, while using much fewer resources with much lower latency. Such consistent results

TABLE II
ACCURACY EVALUATION AND HARDWARE RESOURCE CONSUMPTION FOR AN EXACT MULTIPLIER, PACE AND PRIOR DIVIDERS.

Designs		MAE	MRED	RMSE	Area (μm^2) / ratio	Delay (ns) / ratio	Power (mW) / ratio	Norm. ADP
Synopsys DesignWare	Divider	-	-	-	15143 / 1×	9.54 / 1×	20.15 / 1×	1×
	Multiplier	-	-	-	7574 / 0.50×	4.63 / 0.48×	6.62 / 0.32×	0.242×
Proposed PACE	L1	0.028	0.026	0.045	**1978 / 0.13**×	**2.11 / 0.22**×	**1.23 / 0.06**×	**0.028**×
	L2	0.015	0.014	0.022	2756 / 0.18×	2.48 / 0.25×	1.89 / 0.09×	0.047×
	L3	0.011	0.011	0.014	3275 / 0.21×	2.93 / 0.30×	2.41 / 0.11×	0.066×
	L4	**0.005**	**0.005**	**0.007**	4590 / 0.30×	3.43 / 0.35×	3.58 / 0.17×	0.108×
Prior Approximate Dividers	FaNZeD	0.030	0.028	0.039	**1660 / 0.11**×	**1.47 / 0.15**×	**0.93 / 0.05**×	**0.017**×
	TruncApp	0.085	0.083	0.095	5643 / 0.37×	5.48 / 0.57×	4.61 / 0.22×	0.214×
	LEAD	0.021	0.020	0.028	7814 / 0.51×	7.31 / 0.77×	3.82 / 0.19×	0.395×
	QIAD	**0.006**	**0.005**	**0.006**	13594 / 0.89×	9.89 / 1.04×	11.82 / 0.59×	0.930×

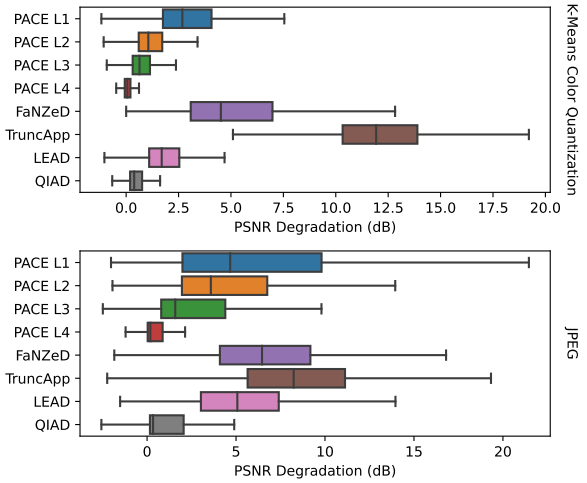

Fig. 7. Statistics of PSNR degradation due to approximate divisions on Caltech101 dataset.

highlight the wide applicability and effectiveness of our PACE divider in various division scenarios.

VI. CONCLUSIONS

In this work, a piece-wise approximate FP divider PACE is proposed. The multi-level architecture seamlessly integrates run-time configurability without significantly increasing area and delay overhead, while inherently achieving a commendable level of accuracy. When compared to the state-of-the-art designs, the proposed divider strikes an advantageous balance between accuracy and resource efficiency. Furthermore, the application-level evaluation attests to the marginal decline in output quality when utilizing the proposed dividers.

ACKNOWLEDGMENT

The authors thank the supports from NSFC (Grant No. 62034007, 62141404 and 62304074), Major Program of Zhejiang Provincial NSF (Grant No. D24F040002), SGC Cooperation Project (Grant No. M-0612) and the Zhejiang University Education Foundation Qizhen Scholar Foundation.

REFERENCES

[1] W. Liu *et al.*, "A retrospective and prospective view of approximate computing," in *Proc. of the IEEE*, vol. 108, no. 3, pp. 394–399, 2020.
[2] C. Wen *et al.*, "Approximate floating-point FFT design with wide precision-range and high energy efficiency," in *ASP-DAC'23*, p. 134-139.
[3] C. Chen *et al.*, "Pam: A piecewise-linearly-approximated floating-point multiplier with unbiasedness and configurability," *IEEE Trans. on Computers*, vol. 71, no. 10, pp. 2473–2486, 2021.
[4] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, 2016.
[5] W. Xiao *et al.*, "OPACT: Optimization of Approximate Compressor Tree for Approximate Multiplier," in *DATE'22*, pp. 178–183.
[6] H. Jiang *et al.*, "Approximate arithmetic circuits: A survey, characterization, and recent applications," in *Proc. of the IEEE*, vol. 108, no. 12, pp. 2108–2135, 2020.
[7] Y. Wu *et al.*, "A survey on approximate multiplier designs for energy efficiency: From algorithms to circuits," *ACM Trans. Des. Autom. Electron. Syst.*, 2023.
[8] S. Vahdat *et al.*, "TruncApp: A truncation-based approximate divider for energy efficient DSP applications," in *DATE'17*, pp. 1635–1638.
[9] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.
[10] T. Haveliwala, "Topic-sensitive pagerank: a context-sensitive ranking algorithm for web search," *IEEE Trans. on Knowledge and Data Engineering*, vol. 15, no. 4, pp. 784–796, 2003.
[11] J. Cortes *et al.*, "Coverage control for mobile sensing networks," *IEEE Trans. on Robotics and Automation*, vol. 20, no. 2, pp. 243–255, 2004.
[12] L. Chen *et al.*, "Design of Approximate Unsigned Integer Non-restoring Divider for Inexact Computing," in *GLSVLSI'15*, pp. 51–56.
[13] S. Hashemi *et al.*, "A low-power dynamic divider for approximate applications," in *DAC'16*, pp. 1–6.
[14] H. Jiang *et al.*, "Low-Power Unsigned Divider and Square Root Circuit Designs Using Adaptive Approximation," *IEEE Trans. on Computers*, vol. 68, no. 11, pp. 1635–1646, 2019.
[15] L. Chen *et al.*, "Design of approximate high-radix dividers by inexact binary signed-digit addition," in *GLSVLSI'17*, pp. 293-298.
[16] J. N. Mitchell, "Computer Multiplication and Division Using Binary Logarithms," *IRE Trans. on Elect. Comput.*, vol. EC-11, no. 4, pp. 512–517, 1962.
[17] H. Saadat *et al.*, "Approximate Integer and Floating-Point Dividers with Near-Zero Error Bias," in *DAC'19*, pp. 1–6.
[18] O. G. Ratnaparkhi and M. Rao, "LEAD: Logarithmic Exponent Approximate Divider For Image Quantization Application," in *GLSVLSI'22*, pp. 437–442.
[19] J. Y. L. Low and Ching Chuen Jong, "Non-iterative high speed division computation based on Mitchell logarithmic method," in *ISCAS'13*, pp. 2219–2222.
[20] M. Imani *et al.*, "CADE: Configurable Approximate Divider for Energy Efficiency," in *DATE'19*, pp. 586–589.
[21] J. Oelund and S. Kim, "ILAFD: Accuracy-Configurable Floating-Point Divider Using an Approximate Reciprocal and an Iterative Logarithmic Multiplier," in *GLSVLSI'23*, pp. 639–644.
[22] R. Zendegani *et al.*, "SEERAD: A High Speed yet Energy-Efficient Rounding-based Approximate Divider," in *DATE'16*, pp. 1481–1484.
[23] J. Melchert *et al.*, "SAADI-EC: A Quality-Configurable Approximate Divider for Energy Efficiency," *IEEE Trans. on VLSI Systems*, vol. 27, no. 11, pp. 2680–2692, 2019.
[24] K. J. N. S. Bhargav *et al.*, "A newton raphson method based approximate divider design for color quantization application," in *ISOCC'21*, pp. 115–116.
[25] H. Liu *et al.*, "QIAD: A quadratic interpolation approximate divider," *IEICE Electronics Express*, vol. 20, no. 11, pp. 20230167, 2023.
[26] F.-F. Li *et al.*, "Caltech 101," Apr 2022.