

SAADI-EC: A Quality-Configurable Approximate Divider for Energy Efficiency

Jackson Melchert, Setareh Behroozi^{1b}, *Student Member, IEEE*, Jingjie Li^{1b}, *Student Member, IEEE*,
and Younghyun Kim^{1b}, *Member, IEEE*

Abstract—Energy efficiency is one of the most crucial constraints that dictate performance, lifetime, form factor, and cost in modern computing system design. However, the energy efficiency improvement driven by semiconductor technology scaling is coming to an end with the prediction of the end of Moore’s law in the near future. Approximate computing is a new paradigm to accomplish energy-efficient computing in this twilight of Moore’s law by relaxing exactness requirement of computation results for intrinsically error-resilient applications, such as some machine learning and signal processing. In this paper, we propose an approximate binary divider design that features dynamic configurability of accuracy and energy consumption. Conventional approximate binary dividers lack runtime energy-quality scalability, which is the key to maximizing energy efficiency while meeting the dynamically varying accuracy requirements of the application. Our divider, named Scalable Accuracy Approximate Divider with Error Compensation (SAADI-EC), supports dynamic energy-quality scalability by the incremental approximation of the reciprocal of the divisor using Taylor series expansion. As a result, the speed and energy efficiency of division can be dynamically traded for accuracy by controlling the number of iterations for the approximation. In addition, SAADI-EC corrects the approximation error using simple, yet effective error compensation hardware to greatly improve the accuracy compared to the base implementation, SAADI. For the 8-bit approximation of 32-bit/16-bit division, the average accuracy of SAADI-EC can be adjusted from 94.2% to 99.6% by varying latency and energy $7\times$. In terms of energy $\times$ delay cost, our design costs up to 87% less than other approximate binary dividers for the same accuracy level. We evaluate the accuracy and energy consumption of SAADI-EC for various design parameters and demonstrate its efficacy for low-power signal processing applications including k -means color quantization, JPEG image compression, and image division for video sequences.

Index Terms—Approximate computing, arithmetic logic unit, energy-quality scaling, low-power system.

Manuscript received February 18, 2019; revised May 21, 2019; accepted June 8, 2019. Date of publication July 19, 2019; date of current version October 23, 2019. This work was supported in part by NSF under Award CNS-1845469, in part by the Wisconsin Alumni Research Foundation (WARF), and in part by the Office of the Vice Chancellor for Research and Graduate Education (OVRGE). The preliminary version of this paper was presented at the Asia and South Pacific Design Automation Conference (ASP-DAC), January 2019. (*Corresponding author: Younghyun Kim.*)

The authors are with the Department of Electrical and Computer Engineering, University of Wisconsin–Madison, Madison, WI 53706 USA (e-mail: jmelchert@wisc.edu; jingjie.li@wisc.edu; sbheerozi@wisc.edu; younghyun.kim@wisc.edu).

Color versions of one or more of the figures in this article are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TVLSI.2019.2926083

I. INTRODUCTION

ENERGY efficiency is a daunting challenge in computing system design, and energy-efficient arithmetic units are key building blocks toward this goal. Researchers have sought to develop fast and low-power arithmetic algorithms and circuit implementations, and, as a result, various designs have been proposed to improve performance and reduce the power consumption of arithmetic operations. In the traditional computing paradigm, arithmetic operations are expected to be always accurate, thus arithmetic units are designed to produce exact results at all times. Although some applications require the arithmetic operation results to be perfectly accurate, some applications are more forgiving and do not need that level of accuracy, depending on the context. For such applications, using always-accurate arithmetic units is waste of energy without any quality gain.

Approximate computing is an emerging computing paradigm to accomplish energy-efficient computing by relaxing accuracy requirements for the applications that do not always demand exact computation results [2]–[4]. Applications, such as machine learning and image processing, are resilient to small computation errors due to the absence of a unique golden outcome, intrinsic noise and redundancy of the input, or the self-healing nature of the algorithm [3]. Approximate computing aggressively exploits the error resiliency by producing just good enough results rather than exact results at much lower power consumption, latency, logic area, or a combination of them. Adopting this broad approach, *approximate arithmetic units* have been proposed to produce approximate arithmetic results that are not exact but good enough for the target application. Many approximate adders [5] and approximate multipliers [6], [7] have been proven to save power and improve performance at the cost of minor accuracy loss.

Division is an arithmetic operation crucial in signal processing. A hardware divider is a costly module in terms of latency and energy consumption due to the high complexity of division algorithms. For example, the integer divider instruction (IDIV) of the AMD 12-h family has a latency of 9–17 cycles for a 16-bit division and 9–25 cycles for a 32-bit division, while the integer multiplier instruction (IMUL) for the same widths takes only three cycles [8]. As another example, on FPGAs, a single-precision floating point divider requires $1.35\times$ – $3\times$ more hardware resources and is 27% slower than the same precision multiplier [9]. Despite the lack of energy-efficient and fast hardware dividers, the design of dividers with these

characteristics has received less attention, mainly due to relatively low utilization compared to other arithmetic operations. However, as distributed signal processing becomes more and more pervasive, the division is playing an increasingly important role in low-power systems, and the need for an energy-efficient divider is increasing. As a result, several approximate divider designs have been recently proposed based on rounding [10], truncation [11], [12], and lookup tables (LUTs) [13].

Unfortunately, these prior approximate dividers support only a single level of accuracy and lack *dynamic quality configurability*. The accuracy level in these designs is determined by a hardware design parameter fixed at design time, such as the size of LUT or the number of truncated bits. Once the design parameter has been fixed, there is no control knob for adjusting the accuracy at runtime, and there is no power/performance benefit from sacrificing accuracy further. However, the accuracy requirement for arithmetic operations is not constant because the impact of approximation to the final quality at the application level is highly input-dependent [14]. Moreover, the application-level quality requirement may change over time. Therefore, approximate computing hardware needs to provide a control knob to the application for the feedback control of the application-level quality requirement [15]. Having no dynamic quality-control knob, the existing approximate dividers need to be designed to meet the most strict accuracy requirement predicted at design time, and as a result, the potential of energy savings and performance improvement is not fully exploited when the actual accuracy requirement is less strict.

In this paper, we propose a novel approximate divider design called Scalable Accuracy Approximate Divider with Error Compensation (SAADI-EC), which is capable of dynamically trading accuracy for latency. SAADI-EC is based on our previous approximate divider design SAADI [1], which finds the approximate reciprocal of the divisor and multiplies it by the dividend to obtain the division result. The iterative reciprocal approximation process, where the accuracy gradually increases as more iterations are performed, enables the dynamic energy-quality tradeoff. The number of iterations can be dynamically adjusted by the application for energy-quality scaling—the application can either obtain high-accuracy division results at the cost of a higher latency and energy consumption or reduce latency and improve energy efficiency by sacrificing accuracy. As compared to its base implementation (SAADI), SAADI-EC has novel error compensation logic to greatly improve the accuracy of the approximate division. To compensate for approximation error of finite-order truncation of Taylor series, the approximate quotient is multiplied by a constant in a LUT indexed by the number of iterations.

In summary, the contributions of this paper are as follows.

- 1) We introduce an approximate divider, named SAADI-EC, which features dynamic energy-quality scalability. SAADI-EC performs division by multiplying the dividend by the approximate reciprocal of the divisor. The accuracy of division can be dynamically controlled by adjusting the number of iterations for reciprocal approximation.
- 2) We introduce an efficient method, which is new in SAADI-EC, that compensates for the negatively biased errors introduced in the reciprocal approximation by finite-order truncations of Taylor series.
- 3) We present a low-power hardware architecture of SAADI-EC and analyze the tradeoff between accuracy and energy efficiency. We evaluate the accuracy and energy consumption of SAADI-EC for different design-time parameters as well as runtime parameters.
- 4) We demonstrate the application of SAADI-EC to: a) k -means clustering, b) JPEG image compression, and c) an image division algorithm for video sequences with dynamic accuracy requirements. We illustrate the superiority of runtime accuracy scaling by comparing the area, energy, and delay of the resulting system with SAADI-EC compared to its base implementation and other approximate dividers.

II. RELATED WORK

In the past, researchers have developed various approximate computing techniques, ranging from approximate logic circuits [16], [17] to approximate algorithms and applications [18], [19]. The notion of approximate computing has been adopted to noncompute components, such as memory [20]–[24], sensors [14], [25], bus interfaces [26]–[29], and wireless communication [30], [31].

To improve the energy efficiency of arithmetic operations, various approximate arithmetic units have been proposed. Approximate arithmetic operations can be realized by scaling down the supply voltage below the level that guarantees the Boolean correctness of the circuit or by implementing only partially correct Boolean logic using a fewer number of transistors [5]. Truncating operands and using a narrow-width arithmetic unit is a widely adopted approach [7], [11], [12].

More recently, approximate multipliers with runtime adjustable accuracy through dynamically adjusting the partial product perforation and rounding have been proposed [32]. These types of designs allow for application dependent energy savings and energy-error tradeoff optimization. Some works deeply explore the energy-accuracy tradeoff curve, highlighting the necessity of multiple configurations of approximate arithmetic blocks for the largest energy savings [33]. In stochastic computing, numbers are represented as a stream of bits which can be processed using a simple AND gate or a multiplexer instead of a binary multiplier or an adder [34], [35].

Approximate dividers have been studied relatively recently compared to other arithmetic units such as adders [5] and multipliers [6], [7]. An approximate divider design presented in [13] calculates the quotient using a 2-D LUT indexed by the operands. The accuracy is determined by the granularity of the LUT and hence is fixed at design time. Another design presented in [11] normalizes the operands to remove leading zeros and takes only a small number of the most relevant bits. The truncated dividend is then divided by the truncated divisor using a narrow-width precise divider. A more recent design, called AAXD [36], compares the magnitudes of the dividend

and the divisor to address the overflow problems of [11]. In both [11] and [36], the accuracy is preset by the divider width, which is not dynamically adjustable. SEERAD [10], TruncApp [12], and PLApp [37] are multiplicative dividers, where the reciprocal of the divisor is obtained first and then subsequently multiplied by the dividend. In SEERAD, the reciprocal is approximated using a table indexed by the upper bits of the divisor, wherein TruncApp, the reciprocal is obtained by simple bit manipulations such as inversion, truncation, and concatenation. In PLApp, the reciprocal is approximated using a piecewise linear function and rounding rather than truncation. In all three designs, the accuracy is determined at design time and not configurable at runtime. In INZeD [38], the error introduced by the approximate divider is corrected in order to minimize the overall error bias. This helps to improve the average accuracy and allows for error cancellation in applications prone to error accumulation. Our proposed design, SAADI-EC, uses error correction to achieve low error bias as well. In summary, to the best of our knowledge, SAADI-EC (and SAADI) is the first approximate divider that supports dynamic accuracy configuration.

III. PROPOSED APPROXIMATE DIVIDER

In this section, we introduce our divider design, SAADI-EC and present detailed hardware implementation. We also introduce our error compensation hardware and analyze the accuracy of the proposed design.

A. Proposed Approximate Division

The proposed design, SAADI-EC, falls into a category of *multiplicative division* algorithms, where the reciprocal of the divisor is obtained first and then multiplied by the dividend. We first describe the basics of multiplicative division [39] before we describe our approach.

Let A and B be the dividend and the divisor, respectively. They can be represented in a normalized form as

$$A = 2^{e_a} \times a \quad \text{and} \quad B = 2^{e_b} \times b \quad (1)$$

where $0.5 \leq a < 1$ and $0.5 \leq b < 1$, and e_a , and e_b represent the position of the leading 1 bit if A and B were represented as binary numbers, respectively. During normalization, a and b are truncated to n bits, where n is the width of multiplier, which is determined at design time. Then, the quotient, Q , of division A/B is

$$Q = \frac{A}{B} = 2^{e_a - e_b} \times \frac{a}{b} = 2^{e_a - e_b} \times a \times R(b) \quad (2)$$

where $R(b)$ is the reciprocal of b , i.e., $R(b) = 1/b$. Instead of computing a/b directly, the multiplicative division first finds $R(b)$ and then multiplies it by a , followed by a shift operation for denormalization. The key to fast division is finding the reciprocal as quickly as possible. However, finding an accurate reciprocal is almost as complex as the division itself. Therefore, it is typical that an approximate reciprocal is obtained initially, and then a functional iteration algorithm, such as the Newton–Raphson method, is used to converge to the accurate reciprocal.

When a moderate-accuracy quotient is good enough, using such a time-consuming functional iteration algorithm may be a waste of time and energy. Approximate multiplicative dividers discussed in Section II obtain an approximation of the reciprocal instead of the exact reciprocal to save energy. However, their fixed-accuracy approximation may not always produce results with just good enough accuracy at minimal power consumption. We address this challenge by accuracy-adaptive reciprocal approximation based on incremental Taylor series expansion. Taylor series expansion is a widely used approach to find an approximate reciprocal. More specifically, the Taylor series expansion of $R(b) = 1/b$ is

$$R(b) = \frac{1}{1+x} = \sum_{i=0}^{\infty} (-x)^i = 1 - x + x^2 - x^3 + x^4 - \dots \quad (3)$$

where $x = b - 1$. This series always converges since $-0.5 \leq x < 0$ for normalized b such that $0.5 \leq b < 1$.

Note that the contribution of each term to accuracy improvement exponentially decreases as the order increases. Typically, a good reciprocal approximation can be obtained by adding up the first few low-order terms, and if higher accuracy is required, more high-order terms can be added to incrementally improve the accuracy. Let $\tilde{R}_t(b)$ denote the t th-order approximation of $R(b)$, such that

$$\tilde{R}_t(b) = \sum_{i=0}^t |x|^i = 1 + |x| + |x|^2 + |x|^3 + |x|^4 + \dots + |x|^t \quad (4)$$

which takes $t - 1$ multiplications. The result is a tradeoff between higher accuracy and longer latency. This tradeoff can be exploited by the application depending on its accuracy requirements and latency constraints. Finally, the t th-order approximate quotient $\tilde{Q}_t$ is then obtained as

$$\tilde{Q}_t = 2^{e_a - e_b} \times a \times \tilde{R}_t(b). \quad (5)$$

B. Error Compensation

To improve the approximation accuracy of the base implementation we introduce novel logic for error compensation in SAADI-EC. As shown in (4), all terms in the reciprocal approximation are positive, meaning the resulting approximation error is always negative for a finite t . Therefore, this error can be mitigated by scaling the quotient of SAADI-EC by a positive compensation factor, c . The new scaled quotient $\tilde{Q}_t^C$ is given by

$$\tilde{Q}_t^C = \tilde{Q}_t \times c. \quad (6)$$

Because $0 < |x| \leq 0.5$, and t is always positive, the minimum possible error for $\tilde{R}_t(b)$ is 0% when $|x|$ is very small and t is very large. Therefore, the minimum value of c is 1, equivalent to no error compensation. The maximum possible error for $\tilde{R}_t(b)$ is 25%, when $|x| = 0.5$ and $t = 1$. Therefore, the maximum value of c is 1.333.

Fig. 1 shows how the approximation error of $\tilde{Q}_t$ decreases as t increases. In addition, as t increases, the error distribution

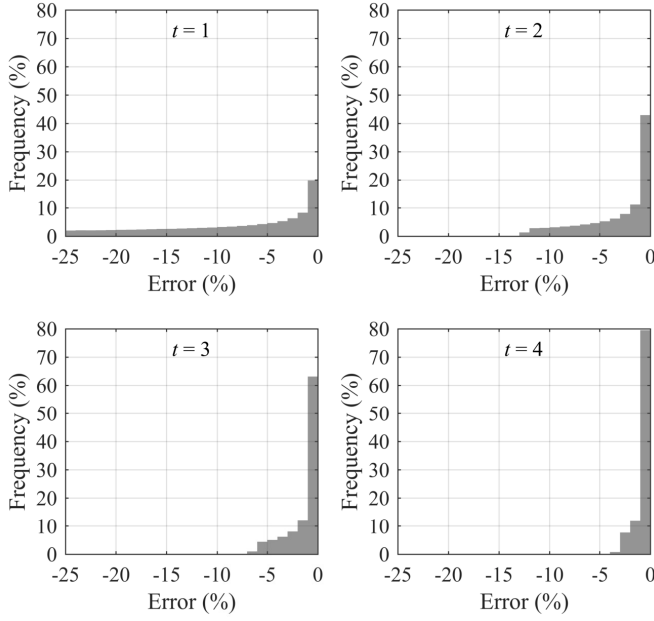
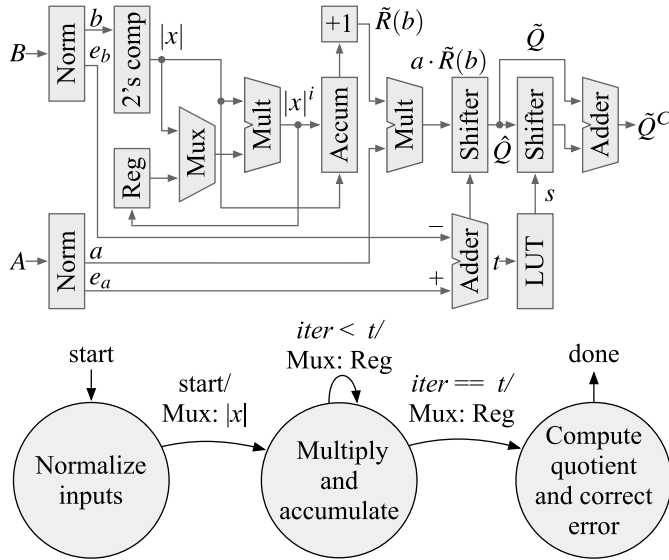
Fig. 1. Error distribution of the approximated quotient with $t = [1, 2, 3, 4]$.

Fig. 2. Hardware architecture and control FSM of SAADI-EC.

gets more closely centered around 0%. For each value of t , the error distribution is different, so a unique scaling factor should be determined at each t to minimize the error. We discuss how this error compensation is implemented without using a costly floating point multiplier in Section III-C.

C. Hardware Architecture

Fig. 2 shows the hardware architecture and a simplified diagram of a finite-state machine (FSM) to control the hardware. Fig. 3 shows the computation flow of SAADI-EC. First, the dividend A and the divisor B are normalized to a and b by the normalizer blocks, respectively, and truncated to n bits. Normalization is done by finding the position of the leading 1

TABLE I
SCALING FACTOR c AND THE RESULTING ERROR COMPENSATION VALUE s FOR ALL VALUES OF t AT $n = [4, 8, 12, 16]$. NOTE THAT $n = 4$ AND $t = 3$ HAS NO ENTRY SINCE THE ACCURACY SATURATES AT $t = 2$

t	Bit width n (bits)							
	4		8		12		16	
t	c	s	c	s	c	s	c	s
1	1.1591	3	1.0634	4	1.0577	4	1.05746	4
2	1.1147	3	1.0219	6	1.0151	6	1.01466	6
3			1.0139	6	1.0042	8	1.00373	8
4			1.0116	6	1.0015	9	1.00097	10
5			1.0106	7	1.0010	10	1.00028	12
6			1.0099	7	1.0009	10	1.00011	13
7			1.0097	7	1.0008	10	1.00007	14
8					1.0008	10	1.00007	14
9					1.0008	10	1.00006	14
10					1.0008	10	1.00006	14
11					1.0007	10	1.00006	14
12							1.00006	14
13							1.00006	14
14							1.00005	14
15							1.00005	14

and left-shifting to make the most significant bit (MSB) 1. If B is a power of 2, the division is done by a simple shift operation. We do not show this in the diagram for simplicity. The 2's complement block converts b to $|x|$. A single-cycle multiplier first computes $|x| \cdot |x| = |x|^2$, and the output is fed back to the multiplier again to subsequently obtain $|x|^3, |x|^4, \dots, |x|^t$ over $t - 1$ cycles. Note that the output of the n -bit multiplier is truncated from $2n$ bits to n bits because it is going to be fed back to itself. The results are added up by the accumulator block in the same cycle to calculate $|x| + |x|^2 + \dots + |x|^t$.

The output of the accumulator is $\tilde{R}_t(b) - 1$, thus it is added to 1 before being multiplied by a .

In the t th cycle, $\tilde{R}_t(b)$ is multiplied by a , and the output is denormalized using a barrel shifter to obtain the approximate quotient $\tilde{Q}$.

Finally, the error compensation discussed in Section III-B is performed. We simplify the error compensation logic by replacing the costly multiplication in (6) by a simple table look-up and an addition. For a given n , which is a design parameter, the appropriate scaling factor c is calculated for $1 \leq t \leq n - 1$ by finding the scaling factor that minimized the mean absolute error (MAE) of $\tilde{Q}_t$ over a large sample of inputs. Error compensation is then done by adding a bit-shifted value of $\tilde{Q}_t$ to itself, i.e.,

$$\tilde{Q}_t^C = \tilde{Q}_t \times c \approx \tilde{Q}_t + \tilde{Q}_t/2^s \quad (7)$$

where the integer bit-shift amount s is given by

$$s = -\text{Round}(\log_2(c - 1)). \quad (8)$$

This simple error compensation hardware is composed of only a shifter, an adder, and an LUT holding the $n - 1$ values of s corresponding to all possible values of t at a given n . The values of c and s for all values of t at $n = [4, 8, 12, 16]$ are given in Table I.

The width of the all the normalized numbers, such as $a, b, |x|, |x|^i$, etc., is n bits. The accumulator needs to be wider

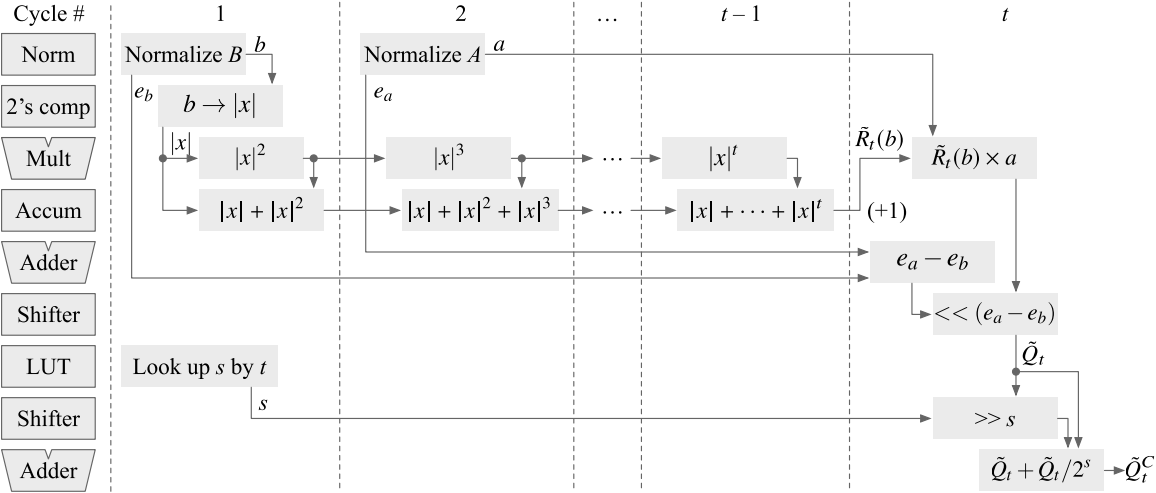


Fig. 3. Hardware block usage and data flow ($t \geq 2$).

than $|x|^i$ since the accumulation can generate a carryout. The maximum value in the accumulator is 1, since the maximum value of $\tilde{R}_t(b)$ is 2 when $|x| = 0.5$. Therefore, the width of the accumulator needs to be only 1 bit wider than the multiplier, thus $n+1$ bits. After adding 1 to the accumulator output, $\tilde{R}_t(b)$ becomes $n+2$ bits, thus it is right-shifted by two bits to fit into the n -bit multiplier.

The reciprocal approximation requires up to $t-1$ multiplications, and $a \times \tilde{R}_t(b)$ requires another multiplication. Therefore, the maximum latency of one division operation is t multiplications. Iterations for reciprocal approximation can be terminated earlier to reduce latency at the cost of accuracy loss, and we discuss this tradeoff in Section III-D.

To save area, the normalization of A and B can be done by a shared normalizer since the normalization of A is required only after $\tilde{R}_t(b)$ is obtained. Similarly, the multiplications for reciprocal approximation and the multiplication of $a \times \tilde{R}_t(b)$ can be done by a single shared multiplier. Therefore, the major hardware components of SAADI-EC include one n -bit normalizer, one n -bit multiplier, one $(n+1)$ -bit accumulator, one $\log_2 n$ -bit adder, two n -bit barrel shifters, one LUT with $n-1$ entries, one $2n$ -bit adder, and a few multiplexers and registers. The usage of the shared hardware blocks is shown in Fig. 3. The area and power consumption of SAADI-EC are evaluated in Section IV-B.

The proposed divider can be extended to compute signed division with the addition of minimal extra hardware. In the normalization blocks, the sign bit of the dividend and divisor need to be stripped off. If the sign bit was a 1 (i.e., negative), then the sign of the operand needs to be inverted using a 2's complement block. At the end of the division, the sign bit is added back to the quotient and the quotient is inverted if necessary.

D. Accuracy and Latency Tradeoff

Unlike prior approximate dividers whose division accuracy is fixed at design time, SAADI-EC provides a flexible energy-quality tradeoff that can be selected by the application

at runtime. When high-accuracy division is required, the application can increase t , and when low-accuracy division is sufficient, it can decrease t to save energy. Although SAADI-EC enables division operations to be performed at the optimal accuracy level, finding the optimal accuracy is highly application-dependent, which is out of the scope of this paper. In this section, we analyze how t affects the error rate and energy consumption.

The loss of accuracy in SAADI-EC is caused by the following factors.

- 1) ϵ_1 : The input operands A and B are truncated to n bits during normalization.
- 2) ϵ_2 : The approximate reciprocal $\tilde{R}_t(b)$ is the sum of a limited number of $|x|^i$ terms.
- 3) ϵ_3 : Each $|x|^i$ term is computed using a statically truncated multiplier that truncates $2n$ -bit results to n bits.
- 4) ϵ_4 : The approximate reciprocal $\tilde{R}_t(b)$ is truncated from $n+2$ bits to n bits before multiplied by a , and the result is truncated to n bits.

The sign of ϵ_1 is input-dependent. Truncating A contributes to negative factors of ϵ_1 since the truncation results in a smaller dividend, while truncating B has the reverse effect, as a smaller divisor produces a larger quotient. The error due to a finite-order approximation (ϵ_2) is the runtime-adjustable error factor that we exploit for accuracy-latency tradeoff such that

$$\epsilon_2 = \tilde{R}_t(b) - R(b) = - \sum_{i=t+1}^{\infty} |x|^i = -|x|^{t+1} - |x|^{t+2} - \dots \quad (9)$$

Note that ϵ_2 is always negative, and the magnitude decreases as t increases. Due to the accuracy loss in the multiplier (ϵ_3), $|x|^i$ eventually becomes zero even if x is nonzero. Note that multiplying $|x| \leq 0.5$ retires at least one bit per multiplication cycle, thus $|x|^i$ becomes zero within n cycles. Therefore, only up to $|x|^{n-1}$, which is obtained at $(n-2)$ th cycle, contributes to $\tilde{R}_t(b)$, hence the effective range of t for accuracy-latency tradeoff is $1 \leq t \leq n-1$, considering one additional cycle at the end for $a \times \tilde{R}_t(b)$. Increasing t beyond $n-1$ only increases latency but does not improve accuracy ($n=4$ is an exception,

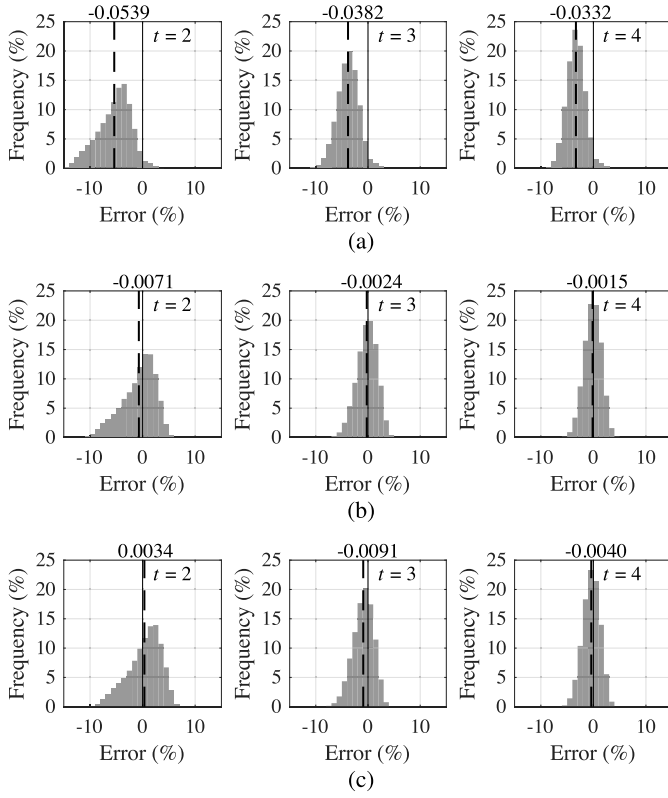


Fig. 4. Error distribution of approximate quotient for varying t with and without error compensation. The dotted line represents the mean error. (a) Before error compensation. (b) After full precision (floating point) error compensation. (c) After proposed error compensation.

where the accuracy saturates at $t = 2$ due to narrow multiplier width). On the other hand, decreasing t below $n - 1$ may terminate the reciprocal approximation for slow-converging $|x|^i$ and increases the average error rate. The error induced by the truncation in the multiplier and the accumulator (ϵ_3 and ϵ_4) is always negative since it results in an under-approximated reciprocal and thus an underapproximated quotient.

The error compensation discussed in Section III-B mitigates the error introduced by ϵ_2 , ϵ_3 , and ϵ_4 . For a given n , a unique scaling factor c is determined for each t and is chosen to minimize the MAE over all inputs. Fig. 4 shows an example of the effect of error compensation on the error distribution of SAADI-EC. In this example, $n = 6$ and $t = [2, 3, 4]$. Without error compensation, Fig. 4(a) shows that resulting error measurements are negatively biased. The mean error becomes closer to 0 as t increases, but the majority of error measurements are negative. Fig. 4(b) shows the error distribution after full precision error compensation according to (6). The error distributions have been shifted so that their mean is much closer to 0. Finally, Fig. 4(c) shows the result of our simplified error compensation according to (7). The error distributions do not vary significantly from the full precision error compensation.

Fig. 5 shows an example of a division operation for $A/B = 190/11$ and $n = 8$. The initial normalization and 2's complement conversion produces $|x| = 0.31250$. After the first iteration, the approximate reciprocal is $\tilde{R}_1(b) = 1.31250$, and the resultant approximate quotient is $\tilde{Q}_1 = 15.5000$.

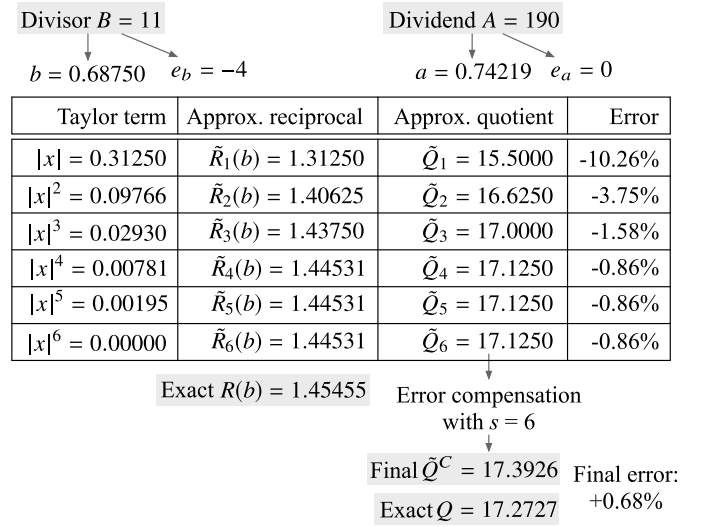


Fig. 5. Example of division operation $190/11$ for $n = 8$.

Since the exact quotient (Q) is 17.2727, the approximation error is -10.26% . As more iterations take place, the accuracy of $\tilde{R}_t(b)$ and $\tilde{Q}_t$ gradually increases. The convergence of $|x|^i$ to zero takes five multiplications, and the error of the quotient after convergence is -0.86% . Note that $\tilde{R}_t(b)$ converges to 1.44531 after three iterations and does not improve further since the $|x|^5$ is already too small and truncated by the accumulator. Therefore, it takes four multiplications (three for $|x|^2$, $|x|^3$, and $|x|^4$, and one for $\tilde{R}_4(b) \times a$) to reach this result. The value of the scaling factor c that minimizes the MAE over all inputs at $n = 8$ and $t = 4$ is 1.0116, and the resulting full-precision error-compensated quotient is $17.1250 \times 1.0116 = 17.3225$. According to (8), this error compensation scaling factor can be approximated as the shift value, s , of 6. Therefore, the error corrected quotient $\tilde{Q}^C$ is $17.1250 + 17.1250/2^6 = 17.3926$, and the final approximation error is 0.68%.

IV. EVALUATION AND DISCUSSION

We evaluate the accuracy and latency of SAADI-EC implemented in Verilog HDL and present the accuracy-latency tradeoff for various design-time and runtime parameters. SAADI-EC is compared with other approximate dividers: SEERAD [10], TruncApp [12], AAXD [36], and PLApp [37]. We also implemented an equivalent MATLAB model of SAADI-EC for application-level quality evaluation: 1) color quantization using k -means clustering; 2) JPEG compression is used as image processing applications, and the results are compared with those of an exact divider; and 3) image division for change detection is then used as an application that has a dynamic accuracy requirement, and the results of this experiment are compared against other proposed approximate dividers.

A. Accuracy and Latency

We first evaluate the arithmetic accuracy of SAADI-EC for various parameters by feeding it one million random combinations of dividends and divisors uniformly distributed within the entire range. As discussed in Section III-D,

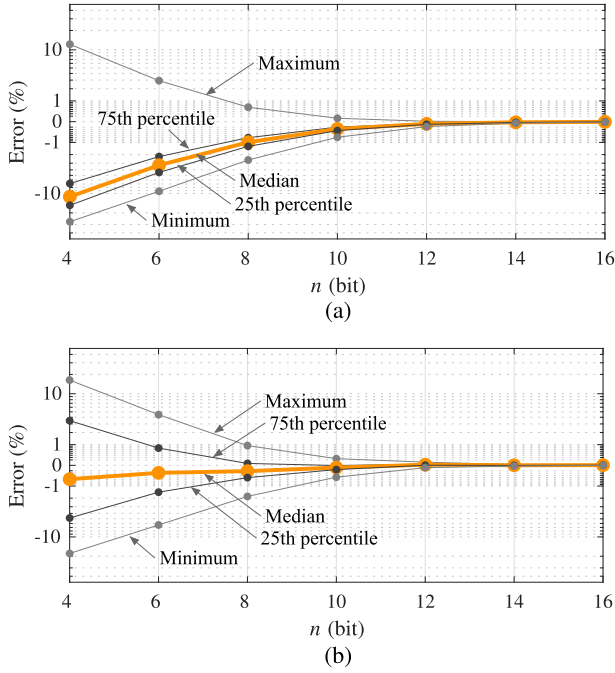


Fig. 6. Error distribution of approximate quotient for varying n with and without error compensation. (a) Before error compensation. (b) After error compensation.

the accuracy of SAADI-EC is determined by a design parameter n , i.e., the width of the truncated operands, and a runtime parameter t , i.e., the order of reciprocal approximation. For a consistent and fair comparison with the previous designs, we assume $2K/K$ division, where the width of the dividend is twice the width of the divisor, and K MSBs of the dividend is smaller than the divisor to prevent overflow [40]. In this paper, we assume 32-bit dividends and 16-bit divisors. Note that this is for a fair comparison, and SAADI-EC is not restricted to any specific dividend-divisor bit ratio.

Fig. 6(a) shows the distribution of approximate division errors before error compensation for $n = [4, 6, 8, \dots, 16]$, while t is set to $n - 1$ to guarantee the convergence of $|x|^i$ to zero. In the figure, each curve shows the maximum, 75th percentile, median, 25th percentile, and minimum error. We can observe that the error rate rapidly decreases as n increases. Overall, as previously discussed in Section III-D, the errors are negatively biased since the combination of the negative error factors ϵ_2, ϵ_3 , and ϵ_4 is more significant than ϵ_1 , which is the only factor that can be positive. Fig. 6(c) shows the result of error compensation on the error distribution. The error distribution is centered much more closely around 0, and the rate of positive errors and negative errors are very similar. Consequently, the maximum and minimum errors are also shifted to a higher value than before error compensation, and the average error and maximum error magnitude decrease across all inputs. The results show that $n \geq 6$ produces very good approximate results with only a few percents maximum error.

Fig. 7(a) illustrates the average number of iterations required to reach the maximum accuracy, up to a maximum of t . As described in Section III-D, the maximum number of effective iterations (that improves accuracy) is $n - 1$, and

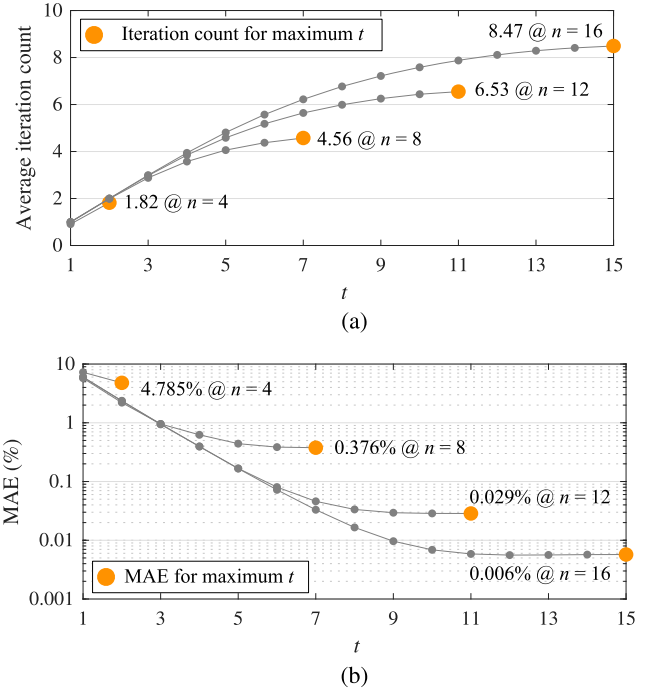


Fig. 7. Average number of iterations and MAE for varying n and t . (a) Average number of iterations for varying n and t . (b) MAE for varying n and t .

TABLE II

AREA, DELAY, POWER, ENERGY PER CYCLE, MINIMUM AND MAXIMUM ACCURACY, AND ENERGY PER OPERATION FOR VARYING n AND ACCURACY REQUIREMENT

Bit width n (bit)		4	8	12	16
Area (μm^2)		1,973	3,035	4,485	6,105
Delay (ns)		1.06	1.22	1.37	1.57
Power (mW)		0.36	0.67	1.28	2.11
Energy per cycle (pJ)		0.39	0.82	1.75	3.31
Min accuracy (%)		92.65	94.23	93.99	93.97
Max accuracy (%)		95.14	99.63	99.97	99.99
Accuracy: 88%	t	1	1	1	1
	Energy (pJ)	0.39	0.82	1.75	3.31
Accuracy: 99%	t	$\times$	3	3	3
	Energy (pJ)	$\times$	2.46	5.25	9.93
Accuracy: 99.9%	t	$\times$	$\times$	6	6
	Energy (pJ)	$\times$	$\times$	10.50	19.86

iteration counts at $t = n - 1$ are highlighted for each n . When $t = n - 1$, it takes about $n/2$ cycles on average to reach the maximum accuracy. This means that the actual average number of iterations to reach the maximum accuracy is smaller than $n - 1$.

Fig. 7(b) shows how MAE varies with t and n for $n = [4, 8, 12, 16]$ and $1 \leq t \leq n - 1$. We observe that, as n increases, not only does the maximum accuracy increase, but the speed of accuracy improvement also increases. As shown in Fig. 7(b), the $n = 8$ curve has a steeper slope than the $n = 4$ curve.

B. Area and Power Consumption

We synthesized the Verilog description of SAADI-EC using Synopsis Design Compiler, targeting the NanGate 45-nm CMOS Technology [41]. We adopt a single-cycle multiplier as the base arithmetic unit. Table II reports the area, delay, power,

TABLE III
COMPARISON WITH PREVIOUSLY PROPOSED APPROXIMATE
DIVIDERS AND AN EXACT DIVIDER

Divider		Cyc.	MAE (%)	Area (μm^2)	Total delay (ns)	Power (mW)	Energy (nJ)
SAADI-EC(4)	Max	2	4.86	1,973	2.12	0.36	0.77
	Min	1	7.35		1.06		0.39
SAADI-EC(8)	Max	7	0.37	3,035	8.54	0.67	5.72
	Min	1	5.77		1.22		0.82
SAADI-EC(12)	Max	11	0.03	4,485	15.07	1.28	19.27
	Min	1	6.01		1.37		1.75
SAADI-EC(16)	Max	15	0.006	6,105	23.55	2.11	49.71
	Min	1	6.03		1.57		3.31
SAADI-EC-P(4)			1	4.86	2,004	0.66	1.18
SAADI-EC-P(8)			1	0.37	9,574	3.05	3.54
SAADI-EC-P(12)			1	0.03	28,797	6.89	9.99
SAADI-EC-P(16)			1	0.006	64,584	13.55	23.58
SAADI(4)	Max	2	11.32	1,199	2.14	0.31	0.66
	Min	1	14.63		1.07		0.33
SAADI(8)	Max	7	0.99	1,963	7.91	0.59	4.67
	Min	1	7.50		1.13		0.66
SAADI(12)	Max	11	0.07	3,068	15.73	1.09	17.22
	Min	1	6.98		1.43		1.56
SAADI(16)	Max	15	0.006	4,872	24.00	1.94	46.76
	Min	1	6.94		1.60		3.11
SEERAD(3)			1	4.41	2,006	1.49	2.76
SEERAD(4)			1	2.12	3,481	2.99	7.33
TruncApp(3)			1	6.37	1,400	0.52	0.70
TruncApp(4)			1	4.26	1,628	0.59	0.94
AAXD(6)			1	6.13	985	0.50	1.15
AAXD(8)			1	2.99	1,386	0.85	2.50
AAXD(10)			1	1.48	1,526	1.02	4.29
AAXD(12)			1	0.74	2,010	1.67	9.02
AAXD(16)			1	0.18	2,432	2.31	21.02
PLApp(4,4)			1	2.82	1,829	0.86	1.08
PLApp(4,5)			1	1.40	2,073	1.15	1.89
PLApp(4,6)			1	0.72	2,383	1.51	2.73
PLApp(8,8)			1	0.18	3,822	2.86	7.21
Exact divider (DesignWare)			1	0	3,206	50.53	95.50

energy per cycle, minimum and maximum accuracy, and energy per operation for different n and accuracy requirements. Some accuracies that cannot be achieved due to the narrow multipliers are indicated with an “ $\times$ ” mark. Note that the same accuracy requirement can be met by multiple designs, but a narrow (low n) design is more energy-efficient than a wide (high n) design. For example, a 99% accuracy can be achieved by 8-, 12-, and 16-bit SAADI-ECs. While the 8-bit SAADI-EC takes the same number of cycles as the 16-bit SAADI-EC (three cycles), the total energy consumption per division is 75% lower.

Table III compares the MAEs of SAADI-EC, SAADI-EC-P (pipelined version of SAADI-EC), SAADI, TruncApp, SEERAD, AAXD, and PLApp with different accuracy levels. The best MAE of TruncApp is 4.26% when the accuracy level is 4, and increasing it beyond 4 does not improve the accuracy due to its linear reciprocal approximation. The MAE of SEERAD(3) and SEERAD(4) is 4.41% and 2.12%, respectively. The MAE of AAXD ranges between 6.13% and 0.18% for accuracy levels 6–16. Although its accuracy is better than TruncApp and SEERAD, its delay is significantly longer

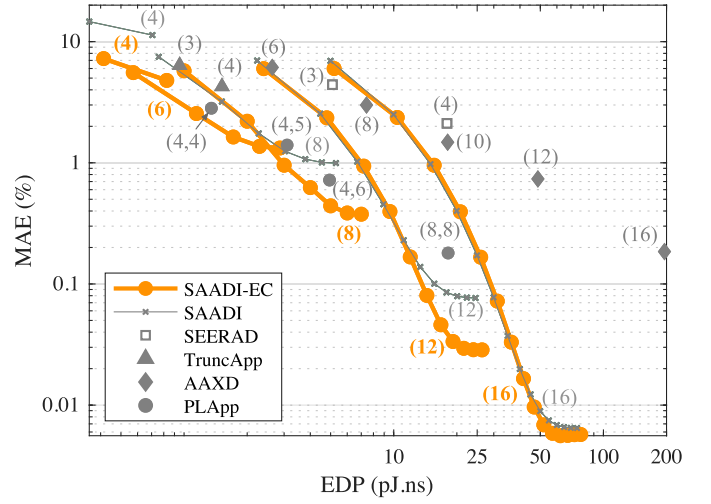


Fig. 8. EDP-MAE tradeoff comparison. TruncApp, SEERAD, AAXD, and PLApp have a single EDP-MAE point per accuracy level. SAADI-EC and SAADI have a flexible EDP-MAE curve per accuracy level.

than the other multiplicative dividers because it uses a divider as the base arithmetic unit. PLApp has two configuration parameters: the first is specifying the number of subintervals used for the reciprocal approximation, while the second is for the rounding width. The MAE of this divider ranges from 2.82% to 0.18%. This design achieves a comparable MAE to AAXD at a much lower delay. The MAE, delay, and energy of SAADI-EC and SAADI are presented by upper bound and lower bound since they are a function of t . Although the previous dividers exhibit lower MAEs for some low accuracy levels, SAADI-EC outperforms others for higher accuracy levels, and, moreover, it is scalable.

We also analyze the performance of SAADI-EC with a pipelined architecture (SAADI-EC-P in Table III). Our basic design can be divided into three stages, one with the normalizing blocks, one with the multiplier and accumulator blocks, and one with the shifting and error compensation blocks. Without pipelining, for a single division operation, all stages need be fully complete before a new operation can be introduced into the first stage, resulting in the under-utilization of hardware as well as a low throughput. To address these problems, multiple copies of the multiply-and-accumulate stage can be introduced to realize pipelined operation as well as hardware to schedule the division operations. This will also improve energy efficiency by keeping the other two stages (normalizing stage and shifting and error compensation stage) fully utilized. In SAADI-EC-P, t is fixed to the maximum value of $n - 1$, and there are $n - 1$ copies of the multiply-and-accumulate pipeline stage. Predictably, the area overhead of this approach is quite large, but the energy per operation decreases for $n = [8, 12, 16]$ compared to SAADI-EC. For example, at $n = 8$ SAADI-EC-P can achieve 0.37% MAE while still producing 1 result per cycle and a delay of 1.16 ns. The energy per result is 3.54 nJ, which is lower than the 5.72 nJ that SAADI-EC consumes.

Fig. 8 better illustrates the comparison between the approximate dividers. It shows the tradeoffs between the energy-delay product (EDP) and accuracy.

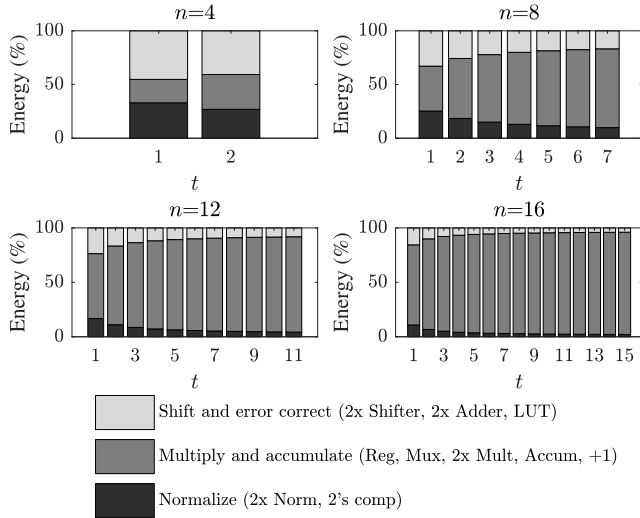


Fig. 9. Energy dissipation breakdown of the main hardware blocks of the proposed divider.

Each curve represents the EDP-accuracy tradeoffs of SAADI-EC for each n , and each dot on the curves represents the EDP-accuracy pair for different t . One can select n based on the range of accuracy requirement and the energy budget at design time, and at runtime, t can be adjusted to exploit the tradeoffs. TruncApp, SEERAD, and PLApp are denoted by “▲,” “□,” and “●” marks, respectively. TruncApp(4), but it is outperformed by SAADI-EC with $n = 6$ and $t = 2$. Similarly, SEERAD(4) is outperformed by SAADI-EC with $n = 8$ and $t = 2$. PLApp is outperformed by SAADI-EC when $n = 6$ and $n = 8$. AAXD is denoted by “◆” marks, and it exhibits high EDP due to the precise divider used as the base arithmetic unit. SAADI-EC shows lower EDP since it uses a multiplier. SAADI-EC outperforms AAXD both in MAE and EDP after a few iterations, which is affordable due to its shorter cycle period.

Fig. 9 reports the sources of energy dissipation within our proposed design as n and t increase. We separated SAADI-EC into three different main blocks: the normalization block, the multiply-and-accumulate block, and the shifting and error compensation block. These represent the logical boundaries between the main sources of energy dissipation. At low values of n , the energy dissipated by the normalization and shift and error compensation blocks is more significant than the multiply-and-accumulate block. As n grows, however, it becomes a significantly larger portion of the energy dissipation in the overall divider, reaching 73.5% of the total energy dissipation when $n = 16$ and $t = 1$. Similarly, as t grows, so does the percentage of energy dissipated by the multiply-and-accumulate blocks, reaching 93.9% of the total energy dissipation when $n = 16$ and $t = 15$.

C. K-Means Color Quantization

For application-level evaluation, we perform color quantization, which reduces the number of unique colors in an image while preserving the appearance of the colors as much as possible. Color quantization saves energy for the storage

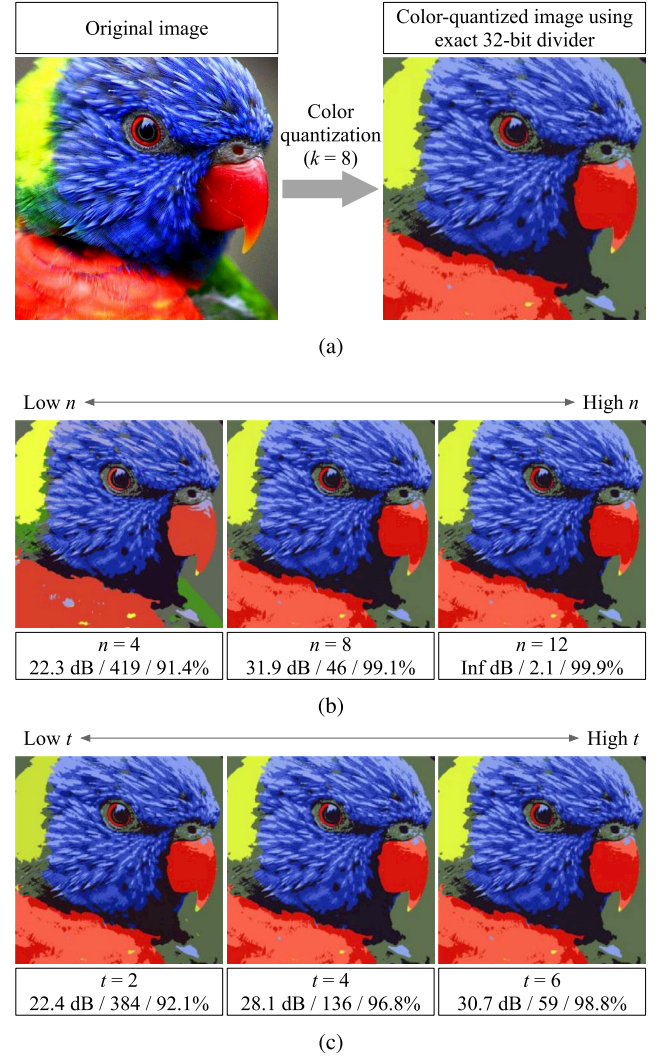


Fig. 10. Color quantization using k -means clustering ($k = 8$) for varying n and t . The output quality is shown in PSNR (dB)/MSE/SSIM (%) in comparison to the output of an 32-bit exact divider. (a) Original image and reference color quantization using an exact divider. (b) Color quantization using SAADI-EC with various n ($t = n - 1$). (c) Color quantization using SAADI-EC with various t ($n = 8$).

or transmission of the image, and k -means clustering is a commonly used algorithm. The mean update operation in k -means requires iterative division by non-constant values, thus an energy-efficient divider needs to be used. To examine the quality of SAADI-EC compared with an exact divider and evaluate the accuracy with varying t and n , five images are clustered into eight colors (i.e., $k = 8$). We calculate three quality metrics: peak signal-to-noise ratio (PSNR), mean square error (MSE), and structural similarity (SSIM), with respect to the baseline results using an exact divider. To evaluate SAADI-EC without the impact of the initial random centers, we feed the same initial centers into the baseline and SAADI-EC each time and repeat the experiment ten times to take the average quality measurements.

Fig. 10(a) shows the original image and the color-reduced image using an exact divider as a reference. Fig. 10(b) shows the results by SAADI-EC with different n . We tie t to $n - 1$, providing sufficient time for SAADI-EC to converge.

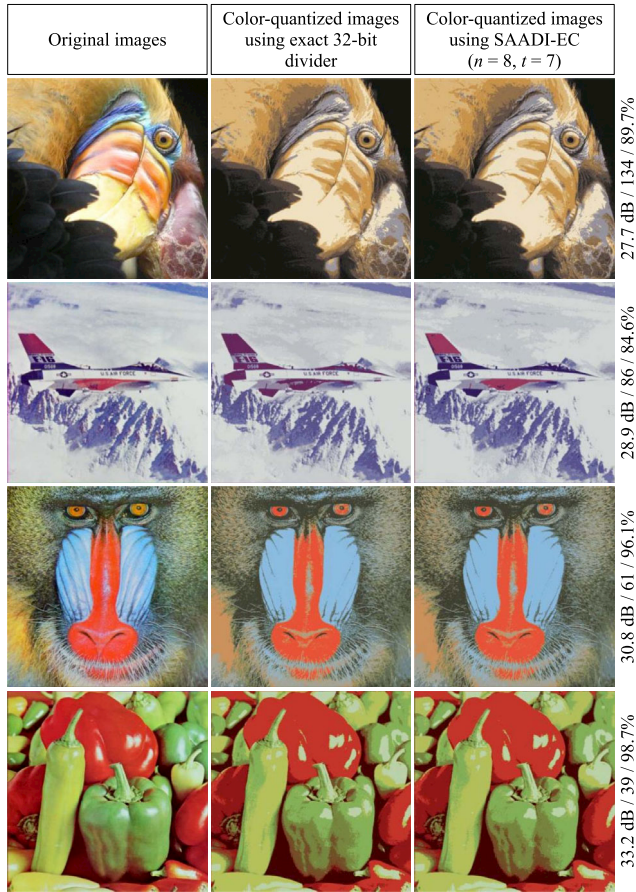


Fig. 11. Color quantization using k -means clustering ($k = 8$) on various images. The output quality is shown in PSNR (dB)/MSE/SSIM (%) in comparison to the output of a 32-bit accurate divider.

As n increases from 4 to 12, all three quality measures improve. Although SAADI-EC is capable of producing an output of a reasonable quality for $n = 4$, the quality dramatically increases for $n = 8$, approaching 100% SSIM for $n = 12$. The results with different t are shown in Fig. 10(c). In this case, n is fixed to 8, and t is set to 2, 4, or 6. We observe a clear trend of quality improvement with a larger t . Even with only two cycles, SAADI-EC produces a high-quality result with SSIM higher than 90%, and further accuracy improvement is achieved by increasing t .

We show the results of color quantization with SAADI-EC for four additional benchmark images in Fig. 11. For all the input images, n is fixed to 8 and t is fixed to 7, and the quality is compared to the output of an exact divider. We can note that the quality measures are dependent on the input image. PSNR ranges from 27.7 to 33.2 dB, and SSIM varies between 84.6% and 98.7%. This confirms the need for a quality-configurable divider, which is the motivation of our work. Exploiting the quality configurability of SAADI-EC, the application can save energy by decreasing t or improve quality by increasing t based on the observed high-level quality.

D. JPEG Compression

To further explore how varying approximation accuracy affects application-level performance, we perform JPEG

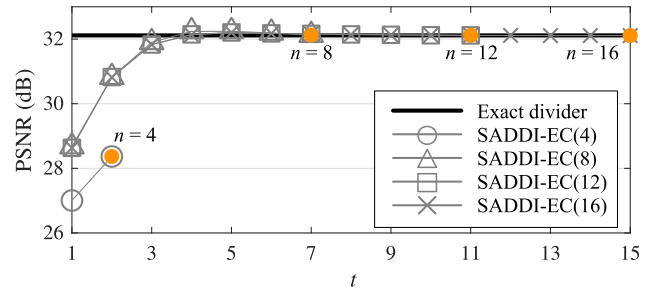


Fig. 12. Image quality (PSNR) using SAADI-EC on JPEG compression.

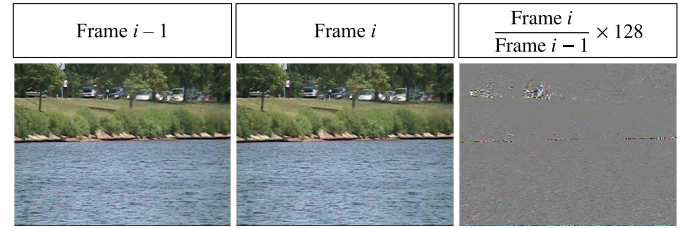


Fig. 13. Example of the image division of two consecutive frames of a benchmark.

compression while varying n and t . The quality of compression for JPEG is set to 90% for this experiment. Our proposed divider replaces the accurate divider used in the quantization step of the JPEG compression algorithm. To evaluate how this substitution affects the quality of the compressed image, we compute the PSNR between the original images and the compressed images with an accurate divider, as well as the PSNR between the original images and the compressed images after substituting our divider. In Fig. 12, we show the results of this application for $n = [4, 8, 12, 16]$ and for t ranging from 1 to $n - 1$. We report the average PSNR over 1000 random images from the Caltech 101 data set [42]. As the quality of quotient approximation increases, the PSNR of the compressed images using SAADI-EC becomes closer to the PSNR of the compressed images using an exact divider. The points along each curve shown in Fig. 12 represent accuracy levels that can be adjusted at runtime. In this particular example, we can see that using 8-bit SAADI-EC suffices.

E. Image Division With Varying Accuracy Requirement

To further illustrate the benefits of dynamic accuracy configuration, we perform image division on video sequences. Image division is a common algorithm used for detecting changes in different frames of a sequence of images, which is useful in applications ranging from surveillance to medical imaging. Fig. 13 shows an example of image division for one of the three benchmarks that we used. In this example, frame i is divided by the previous frame ($i - 1$) and then multiplied by 128. Each pixel in the image is represented by 8-bit integers, so multiplying by 128 allows us to represent the result of the division better within the visible range of the image. Most consecutive frames are very similar, so most pixels in the resulting divided image have a value close to 128. If a pixel in frame i is brighter than in frame $i - 1$, it will appear as a bright spot in the divided image, and if a pixel in

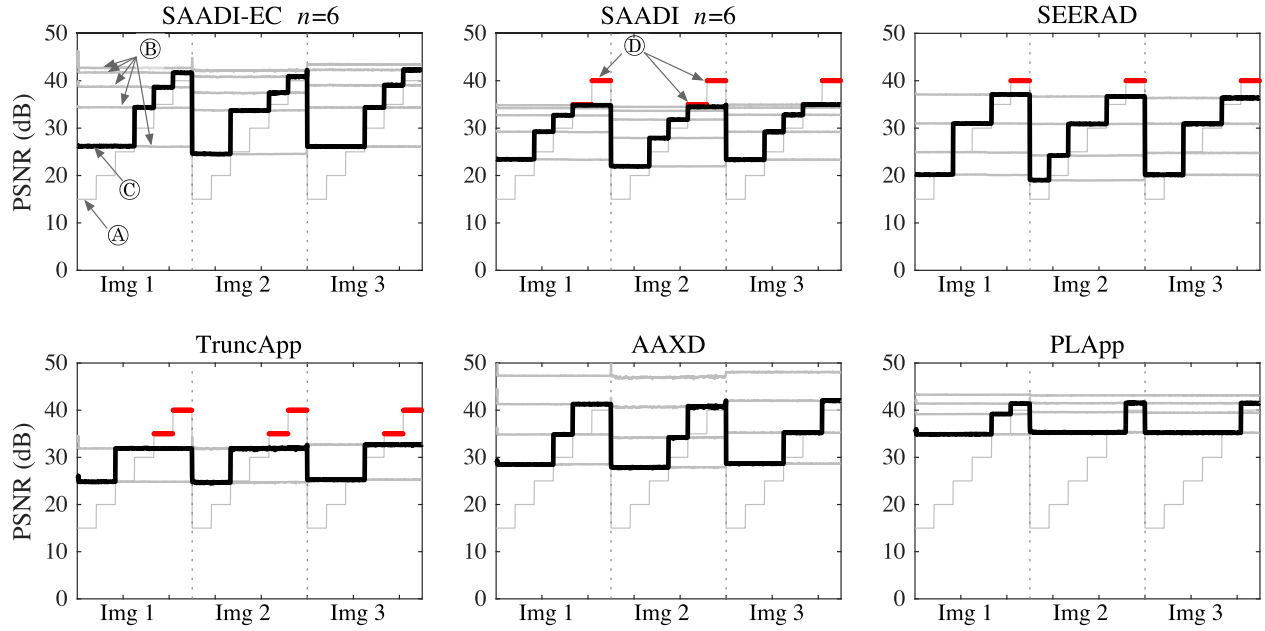


Fig. 14. Resulting PSNR of image division on 1500 frames of change detection benchmarks. For each divider, line “A” denotes the changing accuracy requirement (target PSNR), lines “B” denote accuracy levels supported by the divider, line “C” denote the accuracy level selected by the divider to meet the requirement, and lines “D” denotes violated accuracy requirement (if any).

frame i is darker than in frame $i - 1$, it will appear as a dark spot.

In this work, we simulate the case where the required accuracy of image division varies over time by dividing consecutive frames of a video sequence comprised of three change detection benchmark scenes from [40]. During each of these three scenes, we alter the accuracy requirement of the output image and dynamically adjust the divider to meet the requirement. We evaluate the accuracy of the division periodically throughout the video sequence using a high-accuracy division as a comparison to our approximate division. For SAADI-EC and SAADI, this is done by setting t to the maximum value of 5, which yields almost 100% accurate results. Then, the optimal value of the configuration parameter without violating the accuracy requirement is chosen by gradually decreasing the accuracy and comparing the PSNR between the accurate and approximate divided frames until it drops below the required level.

For evaluation, we measure the accuracy of our divider by comparing it to an accurate divider and calculating the PSNR. Then, we repeat the process with dividers proposed in previous works and compare the results by calculating the number of frames in the sequence that violated the required accuracy level. For the previous dividers, we generously assume that accurate reference division results are obtained without any overhead. Finally, we compare SAADI-EC against previous works by calculating the area, power, and energy required to meet the changing accuracy requirements.

Fig. 14 shows the resulting PSNR and changing accuracy requirements for the three benchmarks. The PSNR is measured and displayed for each of the 500 frames in each benchmark. The different levels of accuracy are calculated by varying the design parameters of each divider. For SAADI-EC and SAADI with $n = 6$, the lowest level of accuracy corresponds

TABLE IV
ACCURACY LEVELS OF SEERAD, TRUNCAPP, AAXD,
AND PLAPP USED FOR COMPARISON

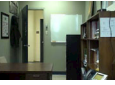
Divider	Less accurate $\longleftrightarrow$ More accurate			
SEERAD(x)	(1)	(2)	(3)	(4)
TruncApp(x)	(3)			(4)
AAXD(x)	(6)	(8)	(10)	(12)
PLApp(x,y)	(4,4)	(2,5)	(4,5)	(2,6)

to $t = 1$, and the highest level of accuracy corresponds to $t = 5$. Note that these accuracy levels are all achievable with a single divider and can be configured at runtime. For SEERAD, TruncApp, AAXD, and PLApp, the accuracy levels correspond to different dividers (B). The accuracy level of these dividers means the bit-width of the base multiplier or divider unit used in each design, and the accuracy levels used in the experiment are listed in Table IV. We assume that, when multiple dividers are used to support dynamic accuracy adjustment, unused dividers are power-gated. The staircase line (A) represents the dynamic target accuracy, and “C” presents the actual accuracy achieved by each divider. Finally, the red segments (D) represent accuracy violations.

Table V illustrates the area, delay, energy, and number of violated frames for each of the three benchmarks as well as the combined total of the benchmarks. SEERAD, TruncApp, AAXD, and PLApp all require multiple dividers to meet the dynamic accuracy requirement. The area is the sum of the area of the dividers used to achieve the target accuracy. [e.g., AAXD(12) is not included in the area since it is not used.] In the case of SEERAD, TruncApp, and the nonerror corrected SAADI, the maximum required accuracy is greater than the maximum accuracy that can be achieved by the divider, therefore there are some violated frames. We observe

TABLE V

AREA, POWER, DELAY, ENERGY, AND NUMBER OF VIOLATED FRAMES FOR THREE CHANGE DETECTION BENCHMARKS

Benchmark image and resolution	Divider	Area (μm^2)	Delay (s)	Energy (mJ)	Violated frames (%)
Image 1					
	SAADI-EC	2,492	2.45	1.20	0
	SAADI	1,490	3.69	1.55	33
	SEERAD	20,617	0.93	3.79	17
	TruncApp	2,744	1.10	0.80	33
	AAXD	3,897	3.41	2.71	0
	PLApp	6,248	1.55	1.55	0
432×288					
Image 2					
	SAADI-EC	2,492	1.64	0.80	0
	SAADI	1,490	2.28	0.95	33
	SEERAD	20,617	0.58	2.39	17
	TruncApp	2,744	0.68	0.50	33
	AAXD	3,897	2.10	1.68	0
	PLApp	6,248	0.91	0.84	0
320×240					
Image 3					
	SAADI-EC	2,492	1.70	0.83	0
	SAADI	1,490	2.56	1.07	33
	SEERAD	20,617	0.64	2.63	17
	TruncApp	2,744	0.73	0.51	33
	AAXD	3,897	2.20	1.65	0
	PLApp	6,248	1.02	0.94	0
360×240					
Total					
Total	SAADI-EC	2,492	5.80	2.83	0
	SAADI	1,490	8.52	3.58	33
	SEERAD	20,617	2.15	8.80	17
	TruncApp	2,744	2.51	1.81	33
	AAXD	3,897	7.72	6.04	0
	PLApp	6,248	3.48	3.32	0

that SAADI-EC outperforms SEERAD, AAXD, and PLApp in terms of area and energy. While TruncApp achieves lower delay and energy than SAADI-EC, its accuracy does not meet the quality requirement.

V. CONCLUSION

An energy-efficient divider is a crucial arithmetic unit for low-power signal processing. In this paper, we presented a quality-configurable approximate divider, named SAADI-EC. It is based on an incremental approximation of the reciprocal of the divisor, where the accuracy gradually increases over multiple iterations. The application can set the number of iterations to exploit the trade-off between accuracy the latency to meet its requirement. We demonstrated that SAADI-EC produces the division results with adjustable accuracy. The accuracy–latency tradeoff of the different implementations of SAADI-EC is evaluated, and the results for 32-bit division with 8-bit approximation show the average accuracy between 94.2% and 99.6% with latency between one and seven cycles. We also performed color quantization using k -means clustering, and JPEG compression and demonstrated that SAADI-EC can produce high-quality results comparable to those generated by an exact divider. Finally, we showed that SAADI-EC outperforms other approximate dividers in applications that have dynamic accuracy requirements using image division.

REFERENCES

- [1] S. Behroozi, J. Li, J. Melchert, and Y. Kim, “SAADI: A scalable accuracy approximate divider for dynamic energy-quality scaling,” in *Proc. Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2019, pp. 481–486.
- [2] J. Han and M. Orshansky, “Approximate computing: An emerging paradigm for energy-efficient design,” in *Proc. 18th IEEE Eur. Test Symp. (ETS)*, May 2013, pp. 1–6.
- [3] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, “Approximate computing and the quest for computing efficiency,” in *Proc. 52nd Annu. Des. Automat. Conf.*, San Francisco, CA, USA, Jun. 2015, pp. 120–120-6.
- [4] M. Alioto, “Energy-quality scalable adaptive VLSI circuits and systems beyond approximate computing,” in *Proc. IEEE DATE*, Lausanne, Switzerland, Mar. 2017, pp. 127–132.
- [5] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, “IMPACT: IMPrecise adders for low-power approximate computing,” in *Proc. 17th IEEE/ACM Int. Symp. Low-Power Electron. Design (ISLPED)*, Aug. 2011, pp. 409–414.
- [6] C. Liu, J. Han, and F. Lombardi, “A low-power, high-performance approximate multiplier with configurable partial error recovery,” in *Proc. Conf. Design, Autom. Test Eur. (DATE)*, Mar. 2014, p. 95:1–95:4.
- [7] S. Hashemi, R. I. Bahar, and S. Reda, “DRUM: A dynamic range unbiased multiplier for approximate applications,” in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2015, pp. 418–425.
- [8] *Software Optimization Guide for AMD Family 10h and 12h Processors*, Adv. Micro Devices, Santa Clara, CA, USA, 2011.
- [9] *Floating-Point IP Cores User Guide*, Altera, San Jose, CA, USA, 2016.
- [10] R. Zendegani, M. Kamal, A. Fayyazi, A. Afzali-Kusha, S. Safari, and M. Pedram, “SEERAD: A high speed yet energy-efficient rounding-based approximate divider,” in *Proc. Conf. Design, Autom. Test Eur. (DATE)*, Mar. 2016, pp. 1481–1484.
- [11] S. Hashemi, R. I. Bahar, and S. Reda, “A low-power dynamic divider for approximate applications,” in *Proc. Design Autom. Conf. (DAC)*, Jun. 2016, pp. 105:1–105:6.
- [12] S. Vahdat, M. Kamal, A. Afzali-Kusha, M. Pedram, and Z. Navabi, “TruncApp: A truncation-based approximate divider for energy efficient DSP applications,” in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Lausanne, Switzerland, Mar. 2017, pp. 1635–1638.
- [13] L. Wu and C. C. Jong, “A curve fitting approach for non-iterative divider design with accuracy and performance trade-off,” in *Proc. IEEE Int. New Circuits Syst. Conf. (NEWCAS)*, Jun. 2015, pp. 1–4.
- [14] A. Raha and V. Raghunathan, “Towards full-system energy-accuracy tradeoffs: A case study of an approximate smart camera system,” in *Proc. Design Autom. Conf. (DAC)*, Jun. 2017, pp. 74:1–74:6.
- [15] V. K. Chippa, S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, “Approximate computing: An integrated hardware approach,” in *Proc. Asilomar Conf. Signals, Syst. Comput.*, Nov. 2013, pp. 111–117.
- [16] J. Park, H. Choo, K. Muhammad, S. Choi, Y. Im, and K. Roy, “Non-adaptive and adaptive filter implementation based on sharing multiplication,” in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, vol. 1, Jun. 2000, pp. 460–463.
- [17] Y. Kim, S. Venkataramani, K. Roy, and A. Raghunathan, “Designing approximate circuits using clock overgating,” in *Proc. Design Autom. Conf. (DAC)*, Jun. 2016, pp. 15:1–15:6.
- [18] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” 2015, *arXiv:1510.00149*. [Online]. Available: <https://arxiv.org/abs/1510.00149>
- [19] S. Han *et al.*, “EIE: Efficient inference engine on compressed deep neural network,” in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit.*, Jun. 2016, pp. 243–254.
- [20] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, “Flicker: Saving DRAM refresh-power through critical data partitioning,” in *Proc. Int. Conf. Architectural Support Program. Lang. Oper. Syst. (ASPLOS)*, Mar. 2011, pp. 213–224.
- [21] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, “Approximate storage in solid-state memories,” *ACM Trans. Comput. Syst.*, vol. 32, no. 3, pp. 9:1–9:23, Sep. 2014.
- [22] A. Ranjan, S. Venkataramani, X. Fong, K. Roy, and A. Raghunathan, “Approximate storage for energy efficient spintronic memories,” in *Proc. Design Autom. Conf. (DAC)*, Jun. 2015, pp. 1–6.
- [23] A. Raha, H. Jayakumar, S. Sutar, and V. Raghunathan, “Quality-aware data allocation in approximate DRAM,” in *Proc. Int. Conf. Compil., Archit., Synth. Embedded Syst. (CASES)*, Oct. 2015, pp. 89–98.

- [24] L. Yang and B. Murmann, "Approximate SRAM for energy-efficient, privacy-preserving convolutional neural networks," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2017, pp. 689–694.
- [25] P. Stanley-Marbell and M. Rinard, "Lax: Driver interfaces for approximate sensor device access," in *Proc. Workshop Hot Topics Oper. Syst. (HotOS)*, May 2015, pp. 1–6.
- [26] P. Stanley-Marbell and M. Rinard, "Reducing serial I/O power in error-tolerant applications by efficient lossy encoding," in *Proc. Design Autom. Conf. (DAC)*, Jun. 2016, pp. 62:1–62:6.
- [27] D. J. Pagliari, E. Macii, and M. Poncino, "Serial T0: Approximate bus encoding for energy-efficient transmission of sensor signals," in *Proc. Design Autom. Conf. (DAC)*, Jun. 2016, pp. 14:1–14:6.
- [28] Y. Kim, S. Behroozi, V. Raghunathan, and A. Raghunathan, "AxSer-Bus: A quality-configurable approximate serial bus for energy-efficient sensing," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, Jul. 2017, pp. 1–6.
- [29] S. Behroozi, V. Raghunathan, A. Raghunathan, and Y. Kim, "A quality-configurable approximate serial bus for energy-efficient sensory data transfer," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 3, pp. 379–390, Sep. 2018.
- [30] S. Sen, S. Schmitt, M. Donahue, and S. Banerjee, "Exploiting 'approximate communication' for mobile media applications," in *Proc. Int. Workshop Mobile Comput. Syst. Appl. (HotMobile)*, Feb. 2009, pp. 11:1–11:6.
- [31] D. Fujiki *et al.*, "High-bandwidth low-latency approximate interconnection networks," in *Proc. Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 469–480.
- [32] V. Leon, G. Zervakis, S. Xydis, D. Soudris, and K. Pekmestzi, "Walking through the energy-error Pareto frontier of approximate multipliers," *IEEE Micro*, vol. 38, no. 4, pp. 40–49, Jul. 2018.
- [33] V. Leon, K. Asimakopoulos, S. Xydis, D. Soudris, and K. Pekmestzi, "Cooperative arithmetic-aware approximation techniques for energy-efficient multipliers," in *Proc. Design Autom. Conf. (DAC)*, Jun. 2019, pp. 160:1–160:6.
- [34] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2s, pp. 92:1–92:19, May 2013.
- [35] S. R. Faraji and K. Bazargan, "Hybrid binary-unary hardware accelerator," in *Proc. Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2019, pp. 210–215.
- [36] H. Jiang, L. Liu, F. Lombardi, and J. Han, "Adaptive approximation in arithmetic circuits: A low-power unsigned divider design," in *Proc. Conf. Design, Autom. Test Eur. (DATE)*, Mar. 2018, pp. 1411–1416.
- [37] M. Vaeztourshizi, M. Kamal, A. Afzali-Kusha, and M. Pedram, "An energy-efficient, yet highly-accurate, approximate non-iterative divider," in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Design (ISLPED)*, Jul. 2018, pp. 14:1–14:6.
- [38] H. Saadat, H. Javadi, and S. Parameswaran, "Approximate integer and floating-point dividers with near-zero error bias," in *Proc. Design Autom. Conf. (DAC)*, Jun. 2019, pp. 161:1–161:6.
- [39] S. F. Oberman and M. J. Flynn, "Division algorithms and implementations," *IEEE Trans. Comput.*, vol. 46, no. 8, pp. 833–854, Aug. 1997.
- [40] B. Parhami, *Computer Arithmetic*. London, U.K.: Oxford Univ. Press, 1999.
- [41] Nangate 45 nm Open Cell Library, NanGate, Santa Clara, CA, USA, 2008.
- [42] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories," in *Proc. CVPR Workshop Generative-Model Based Vis. (WGMVBV)*, Apr. 2004, p. 178.



Jackson Melchert received the B.S. degree in computer engineering and computer science from the University of Wisconsin–Madison, Madison, WI, USA, in 2019.

He has authored and contributed to papers published in ACM Great Lakes Symposium for Very Large Scale Integration (GLSVLSI), the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI), and Asia and South Pacific Design Automation Conference (ASP-DAC). His current research interests include approximate computing,

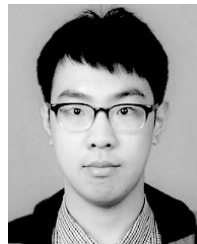
low-power embedded systems, very large-scale integrated circuit design, and computer architecture.



Setareh Behroozi (S'18) received the B.S. degree in computer engineering from the Iran University of Science and Technology, Tehran, Iran, in 2013, and the M.S. degree in computer engineering, computer architecture from the Sharif University of Technology, Tehran, in 2015. She is currently working toward the Ph.D. degree in electrical and computer engineering at the University of Wisconsin–Madison, Madison, WI, USA.

Her current research interests include approximate computing, low-power hardware-software for embedded systems, emerging memory technologies, and computer architecture.

Ms. Behroozi was a recipient of the Electrical and Computer Engineering Chancellor's Opportunity Fellowship, the CRA-W Grad Cohort for Women Workshop, the Grace Hopper Celebration (GHC), the A. Richard Newton Young Student Fellowships, and the Design Contest Award at the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED) in 2018.



Jingjie Li (S'16) received the B.S. degree in electronic information engineering from the Beijing Institute of Technology, Beijing, China, in 2017 and the B.Eng. (Research and Development) degree (Hons.) in electronic and communication systems from The Australian National University, Canberra, ACT, Australia, in 2017. He is currently working toward the Ph.D. degree at the Department of Electrical and Computer Engineering, University of Wisconsin–Madison (UW–Madison), Madison, WI, USA.

From 2016 to 2017, he was a Student Scholar at Data61, Commonwealth Scientific and Industrial Research Organization, Sydney. He currently serves as a Chancellor's Opportunity Fellow at UW–Madison. His current research interests include pervasive computing, Internet of Things, and low-power design for embedded systems.

Mr. Li was a recipient of the A. Richard Newton Young Student Fellowship, the Design Contest Award at the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED) in 2018, and the Best Paper Award of the ACM Conference on Human Factors in Computing Systems (CHI) in 2019.



Younghyun Kim (M'13) received the B.S. degree (Hons.) in computer science and engineering and the Ph.D. degree in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2007 and 2013, respectively.

From 2013 to 2016, he was a Post-Doctoral Research Assistant at the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA. From 2009 to 2011, he was a Visiting Scholar at the University of Southern California, Los Angeles, CA, USA. He is currently an Assistant Professor at the Department of Electrical and Computer Engineering, University of Wisconsin–Madison, Madison, WI, USA. He has coauthored 1 book, 2 book chapters, and more than 70 journal and conference papers. His current research interests include energy-efficient computing, cyber-physical systems security, and the Internet-of-Things.

Dr. Kim was a recipient of the NSF Faculty Early Career Development Program (CAREER) Award in 2019, the EDAA Outstanding Dissertation Award in 2013, the Design Contest Award at the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED) in 2007, 2012, 2017, and 2018, the IEEE SSCS Seoul Chapter Award at the International SoC Design Conference in 2009, and the Best Paper Award Nomination at the ACM/IEEE ISLPED in 2016. He has served on the Technical Program Committees of the ACM/EDAC/IEEE Design Automation Conference (DAC), the ACM/IEEE ISLPED, the Asia and South Pacific Design Automation Conference (ASP-DAC), the International Conference on VLSI Design (VLSID), the Symposium on Applied Computing, and the Ph.D. Forums at DAC and Design Automation and Test in Europe (DATE). He served as a Guest Editor for a Special Issue of *VLSI Integration Journal* (Elsevier).