Approximate Arithmetic Circuits: A Survey, Characterization and Recent Applications

Honglan Jiang, *Member, IEEE,* Francisco Javier Hernandez Santiago, Hai Mo, Leibo Liu*, *Senior Member, IEEE,* and Jie Han*, *Senior Member, IEEE*

***Abstract*—Approximate computing has emerged as a new paradigm for high-performance and energy-efficient designs of circuits and systems. For the many approximate arithmetic circuits proposed, it has become critical to understand a design or approximation technique for a specific application to improve performance and energy efficiency with a minimal loss in accuracy. This article aims to provide a comprehensive survey and a comparative evaluation of recently developed approximate arithmetic circuits under different design constraints. Specifically, approximate adders, multipliers and dividers are synthesized and characterized under optimizations for performance and area, respectively. The error and circuit characteristics are then generalized for different classes of designs. The applications of these circuits in image processing and deep neural networks indicate that the circuits with lower error rates or error biases perform better in simple computations such as the sum of products, whereas more complex accumulative computations that involve multiple matrix multiplications and convolutions are vulnerable to single-sided errors that lead to a large error bias in the computed result. Such complex computations are more sensitive to errors in addition than those in multiplication, so a larger approximation can be tolerated in multipliers than in adders. The use of approximate arithmetic circuits can improve the quality of image processing and deep learning in addition to the benefits in performance and power consumption for these applications.**

***Index Terms*—approximate computing, arithmetic circuits, adders, multipliers, dividers, image processing, deep neural networks.**

I. Introduction

With the increasing importance of big data processing and artificial intelligence, an unprecedented challenge has arisen due to the massive amounts of data and complex computations required in these applications. Energy-efficient and high-performance general-purpose compute engines, as well as application specific integrated circuits, are highly demanded to facilitate the development of these new technologies. Meanwhile, exact or high-precision computation is not always necessary. Instead, some small errors can compensate each other or will not have a significant effect in the computed results. Hence, approximate computing (AC) has emerged as a new approach to energy-efficient design, as well as to increasing the performance of a computing system, at a limited loss in accuracy [1].

A. Motivation

In the past few decades, the feature size of transistors has decreased exponentially, as governed by Moore's law [2], which has resulted in a continuous improvement in the performance and power efficiency of integrated circuits. However, at the nanometer scale, the supply voltage cannot be further reduced, which has led to a significant increase in power density. Thus, a percentage of transistors in an integrated circuit must be powered off to alleviate the thermal issues; the powered-off transistors are called "dark silicon" [3]. A study has shown that the area of "dark silicon" may reach up to more than 50% for an 8 nm technology [4]. This indicates an increasing challenge to improve circuit performance and power efficiency by using conventional technologies. New design methodologies have been investigated to address this issue, including multicore architectures, heterogeneous integration and AC [5].

AC is driven by the observation that many applications, such as multimedia, recognition, classification, and machine learning, can tolerate the occurrence of some errors. Due to the perceptual limitations of humans, some errors do not impose noticeable degradation in the output quality of image, audio and video processing. Moreover, the external input data to a digital system are usually noisy and quantized, so there is already a limit in the precision or accuracy in representing useful information. Probability-based computing such as stochastic computing performs arithmetic functions on random binary bit streams using simple logic gates [6], where trivial errors do not result in a significantly different result. Lastly, many applications including machine learning are based on iterative refinement. This process can attenuate or compensate the effects of insignificant errors [7]. AC has thus become a potentially promising technique to benefit a variety of error-tolerant applications.

B. Development History of Approximate Arithmetic Circuits

Since the 1960s, the Newton-Raphson algorithm has been utilized for computing an approximate quotient to speed up division [8], followed by many other functional iteration-based algorithms such as Goldschmidt [9]. Multiple-precision dividers can, therefore, be obtained by terminating the computing process at different stages [10].

Also in the early 1960s, Mitchell proposed a logarithm-based algorithm for multiplication and division [11]. Although specific approximation techniques aimed at arithmetic circuits were not significantly developed in the following few decades,

*Corresponding authors.

H. Jiang and H. Mo are with the Institute of Microelectronics, Tsinghua University, Beijing, 100084, China.
E-mail: jianghonglan@mails.tsinghua.edu.cn, moh19@mails.tsinghua.edu.cn

L. Liu is with the Institute of Microelectronics, Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing, 100084, China.
E-mail: liulb@tsinghua.edu.cn

F. J. H. Santiago was with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada. Current address: Intel, Zapopan, Mexico.
E-mail: fh@ualberta.ca

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

some straightforward approximation (or rounding) techniques have gradually been considered, for example, in truncation-based multipliers to obtain an output with the same bit width as the inputs. This type of multipliers is referred to as a fixed-width multiplier. The approximation is obtained by accumulating some most significant partial products, along with a correction constant obtained by a statistical analysis as an approximation for the sum of the least significant partial products [12], [13].

In 2004, the concept of approximation was applied to adders and Booth multipliers in a superscalar processor to increase the clock frequency of a microprocessor [14]. The approximate adder is designed by observing that the effective carry chain of an adder is much shorter than the full carry chain for most inputs. On average, the longest carry chain in an n-bit addition is no longer than the binary logarithm of n, or $\log(n)$, as discussed by Burks, Goldstine and von Neumann [15]. Thus, a carry in an n-bit adder is obtained from its previous k input bits rather than all of the previous bits (so, $k < n$). Compared to an accurate adder, the critical path of this approximate adder is significantly shorter. The approximate adder was suggested for use in the generation of the $3\times$ multiplicand required in the radix-8 encoding algorithm for a Booth multiplier.

Since around 2008, approximate adders and multipliers have received significant attention, resulting in various designs; the early ones include the almost correct adder (ACA) [16], the error-tolerant adder [17], the lower-part-OR adder (LOA) [18], the equal segmentation adder (ESA) [19], the approximate mirror adder [20], the broken-array multiplier (BAM) [18], the error tolerant multiplier (ETM) [21] and the underdesigned multiplier (UDM) [22]. In addition, logic synthesis methods have been developed to reduce the circuit area and power dissipation for a given error constraint [23], [24]. Automated processes have also been considered to generate approximate adders and multipliers [25], [26]. Moreover, various computing and memory architectures have been proposed to support AC applications [27], [28]. Especially, a programming language can support approximate data types for low-power computing [29]. Recent, approximate designs include those for dividers [30]–[33], multiply-and-accumulate (MAC) units [34], squaring circuits [35]–[38], square root circuits [39], and a coordinate rotation digital computer (CORDIC) [40].

C. Applications of Approximate Computing

AC has been considered for many applications with error resilience, such as image processing and machine learning, for a higher performance and energy efficiency [41]–[48].

The approximation techniques at algorithm, architecture and circuit levels have been synergistically applied in the design of an energy-efficient programmable vector processor for recognition and data mining [41]. This design achieves an energy reduction of 16.7%-56.5% compared to a conventional one without any quality loss, and 50.0%-95.0% when the output quality is insignificantly reduced.

As basic image processing applications, sharpening, smoothing and multiplication have been used to assess the quality of approximate adders and unsigned multipliers [49]–[51]. Image compression algorithms have been considered for evaluating approximate signed multipliers [43], [44].

Approximate adders and multipliers have been integrated in deep learning accelerators for reducing delay and saving energy [42], [45]–[47], [52]. In [42], truncated 16-bit multipliers with constant error compensation are used in lieu of 32-bit floating-point multipliers in an accelerator for large-scale convolutional neural networks (CNNs) and deep neural networks (DNNs). Up to 83.6% and 86.4% reductions in area and power consumption have respectively been achieved. Designs with various error and circuit characteristics have also been exploited in reconfigurable systems to enhance the reconfiguration flexibility. In [45], [46], approximate adders and multipliers with various levels of accuracy are integrated in a coarse-grained reconfigurable array for different configurations determined on-the-fly by the application requirements. In this way, different performance and energy improvements can be obtained by trading off various levels of processing quality.

In the implementation of a state-of-the-art wireline transceiver, an approximate multiplier is used for low-power digital signal processing [48]. Compared to the accurate design, power is reduced by 40% and the maximum performance is improved by 25%.

D. Scope of This Article

Recent research on AC has spanned from algorithms to circuits and programming languages [51], [53]–[56]. This article aims to provide an overview of approximate arithmetic circuits, various design methodologies, an evaluation and characterization of approximate adders, multipliers and dividers with respect to accuracy and circuit measurements. Three image processing applications and a CNN are implemented to show the capability and performance advantage of approximate arithmetic circuits.

Some preliminary results have been presented in [51], [57]; however, this article presents the following new, distinctive contributions. Instead of undergoing a generic synthesis process, approximate circuits are synthesized and optimized for delay and area, respectively. The results can be used to guide the selection of appropriate designs for an application specific requirement (e.g., high performance or low power). In addition, hardware efficiency and accuracy are jointly considered to show the hardware improvements at the cost of a certain loss of accuracy. Furthermore, a larger class of approximate adders, multipliers and dividers including many recent designs are evaluated; in particular, approximate dividers are extensively analyzed and characterized in detail. Finally, image compression and a DNN are implemented for assessing the quality of approximate adders and signed multipliers to obtain insights into the application of approximate arithmetic circuits in image processing and artificial intelligence systems.

This article is organized as follows. Section II briefly reviews the design methodologies and evaluation metrics. The approximate adders, multipliers and dividers are then presented, synthesized and comparatively evaluated in Sections III, IV and V, respectively. Section VI presents the applications. Finally, Section VII concludes this article and discusses current challenges and future prospects.

II. Background

A. Design Methodologies

An approximate arithmetic circuit can be obtained by using the voltage overscaling (VOS) technique [58]–[60], redesigning a logic circuit into an approximate one [51], and using a simplification algorithm [11].

Using VOS, a lower supply voltage is provided to efficiently reduce the power consumption of a circuit, without having to change the circuit structure. However, a reduced voltage increases the critical path delay, possibly resulting in timing errors [58]. Thus, the output can be erroneous due to the violated timing constraint. Moreover, the error characteristics of such an approximate operation are nondeterministic, as affected by parametric variations [61]. When the most significant bits (MSBs) are affected, the output error can be large [62].

More commonly, an approximate design is derived from an accurate circuit by modifying, removing, or adding some elements. For instance, some transistors in a mirror adder are removed to implement a low-power and high-speed full adder [20]. In addition, an approximate circuit can be obtained by simplifying the truth table or Karnaugh Map (K-Map) [22], [63]. This method results in circuits with deterministic error characteristics. Due to the same structure and design principles, however, the hardware improvements are hardly significant especially when a high accuracy is required.

Compared to addition and subtraction, multiplication, division and square root computation are more complex. Therefore, their functions can be converted to some simpler operations. Mitchell's binary logarithm-based algorithm enables the utilization of adders and subtractors to implement multipliers and dividers, respectively [11]. It is the origin of most current simplification algorithms for approximate multiplier and divider design [30], [64], in parallel with the functional iteration-based algorithms for divider design [10]. By using algorithmic simplification, the performance and energy efficiency of an arithmetic circuit can be significantly improved because of the simplification in the basic circuit structure. Nevertheless, the accuracy of such a design is relatively low; many peripheral circuits are required to achieve a high accuracy, which may limit the hardware efficiency.

Practically, several approximation techniques are often simultaneously utilized in a hybrid approximate circuit [65].

B. Evaluation Metrics

Both error characteristics and circuit measurements need to be considered for approximate circuits.

1) Error Characteristics: Various design metrics and analytical approaches are useful for the evaluation of approximate arithmetic circuits [49], [66]–[73]. Monte Carlo simulation is widely employed to acquire data for analysis. The following metrics have been used to assess the error characteristics.

Two basic error metrics are the error rate (ER) and error distance (ED). The ER indicates the probability that an erroneous result is produced. The ED shows the arithmetic difference between the approximate and accurate results. Given the approximate and accurate results M' and M, respectively, the ED is calculated by $\mathrm{ED} = |M' - M|$. Additionally, the relative error distance (RED) shows the relative difference with respect to the accurate result, given by $\mathrm{RED} = \left|\frac{\mathrm{ED}}{M}\right|$. ED and RED reveal two important features of an approximate design. For two input combinations leading to the same ED, the one that produces a smaller accurate result, M, would result in a larger RED. As the average values of all obtained EDs and REDs, the mean error distance (MED) and mean relative error distance (MRED) are often used to assess the accuracy of an approximate design. They are given by

$$\mathrm{MED} = \sum_{i=1}^{N} \mathrm{ED}_i \cdot P(\mathrm{ED}_i), \tag{1}$$

and

$$\mathrm{MRED} = \sum_{i=1}^{N} \mathrm{RED}_i \cdot P(\mathrm{RED}_i), \tag{2}$$

where N is the total number of considered input combinations for a circuit, and ED_i and RED_i are the ED and RED for the i^{th} input combination, respectively. $P(\mathrm{ED}_i)$ and $P(\mathrm{RED}_i)$ are the probabilities that ED_i and RED_i occur, which are also the probability of the i^{th} input combination. The NMED is defined as the normalization of MED by the maximum output of the accurate design, useful in comparing the error magnitudes of approximate designs of different sizes.

The mean squared error (MSE) and root-mean-square error (RMSE) are also widely used to measure the arithmetic error magnitude. They are computed by

$$\mathrm{MSE} = \sum_{i=1}^{N} \mathrm{ED}_i^2 \cdot P(\mathrm{ED}_i), \tag{3}$$

and

$$\mathrm{RMSE} = \sqrt{\mathrm{MSE}}. \tag{4}$$

In addition, the error bias is given by the average error that is the mean value of all possible errors ($M' - M$). The normalized average error is commonly considered as the average error divided by the maximum output of the accurate design.

Last but not the least, the worst-case error of an approximate circuit reflects the largest ED possible. Generally, it is normalized by the maximum accurate result.

2) Circuit Measurements: The basic circuit metrics include the critical path delay, power dissipation and area. Some compound metrics include the power-delay product (PDP), area-delay product (ADP), and energy-delay product (EDP).

Electronic design automation (EDA) tools are indispensable for circuit implementation and measurement. In general, the circuit is measured based on different process technologies and component libraries, e.g., the 45 nm open source NanGate [74], the 45 nm predictive technology model (PTM) [75], 28 nm CMOS, and 15 nm FinFET models. The configurations for the supply voltage, temperature and optimization options also affect the simulation results. For a fair comparison, the same configurations should be used for different designs.

Conventionally, high performance and power efficiency are respectively pursued as independent design considerations. For instance, to cope with aging-induced timing errors, approximate adders and multipliers are developed for a high speed [76]. High-performance arithmetic circuits are also preferred in real-time machine learning systems [77]. For mobile and embedded devices, however, power-efficiency is key to the extended use of a limited battery life.

In this article, approximate circuits are evaluated for maximizing performance (through delay) or minimizing power (through area). Specifically, approximate designs are implemented in hardware description languages (HDLs) and synthesized using the Synopsys Design Compiler (DC, 2011.09 release) in ST's 28 nm CMOS technology, with a supply voltage of 1.0 V at a temperature of 25°C. To compare speed and power, the approximate circuits are synthesized under different constraints. The critical path delay of a design is set to the smallest value without incurring a timing violation for the delay-optimized synthesis, whereas the area is minimized for the area-optimized synthesis. The DesignWare library and "ultra compile" are used in the synthesis for optimization. The critical path delay and area are reported by the Synopsys DC. Power dissipation is measured by the PrimeTime-PX tool with 10 million random input combinations. As widely used EDA tools in industry and academia, Synopsys DC and PrimeTime-PX provide estimations of timing, area and power dissipation with a prediction error of less than 10% compared with physical implementations [78].

3) Comprehensive Measurements: To provide an overall evaluation of an approximate circuit, both error and circuit characteristics must be considered. Several figures of merit (FOMs) have been developed by combining some error and circuit metrics in an analytical form [79], [80]. However, these FOMs are heuristic-based and lead to different comparison results. In this work, therefore, the delay, power and PDP of approximate circuits are directly compared with respect to their ERs, NMEDs and MREDs.

III. Approximate Adders

A. Preliminaries

An adder performs the addition of two binary numbers and is one of the fundamental arithmetic circuits in a computer. Two basic designs are the ripple-carry adder (RCA) and the carry lookahead adder (CLA). An n-bit RCA consists of n full adders (FAs) connected in series, each of which generates a sum (s_i) and a carry-out (c_{i+1}) by implementing $s_i = a_i \oplus b_i \oplus c_i$ and $c_{i+1} = a_i b_i + a_i c_i + b_i c_i$, where a_i and b_i are the i^{th} least significant bits (LSBs) of the two inputs, c_i is the carry-in, $i = 0, 1, \cdots, n-1$. In an n-bit RCA, the carry of each FA propagates to the next FA, thus the delay and circuit size increase proportionally with n, denoted by $O(n)$. An n-bit CLA consists of n units in parallel; each unit produces the signals of a generate ($g_i = a_i b_i$), a propagate ($p_i = a_i \oplus b_i$) and a sum, where the former two signals are used for generating the lookahead carries. In a CLA, the carry is computed by $c_{i+1} = g_i + p_i c_i = g_i + p_i(g_{i-1} + p_{i-1}c_{i-1}) = \cdots = \sum_{j=0}^{i} g_j \prod_{k=j+1}^{i} p_k + c_0 \prod_{k=0}^{i} p_k$. The delay of a CLA is approximately logarithmic in n, or $O(\log(n))$, which is significantly shorter than the delay of an RCA. However, a CLA requires a larger circuit area (in $O(n\log(n))$), so it incurs a higher power dissipation.

For an adder with a width equal to or larger than 32 bits, the simple carry lookahead structure of CLA is not very efficient due to the large fan-in and fan-out of the constituent gates that lower the speed and increase the circuit area and power consumption. Thus, multiple levels of lookahead structures have been proposed to construct a large-width adder, which is usually referred to as a parallel-prefix adder. The parallel-prefix adders exploit the fact that the carry signals in a CLA can be generated by grouping g_i and p_i in various ways. By varying the group size and the connection pattern, many parallel-prefix adders have been designed to improve the speed or reduce the circuit area, including the Kogge-Stone adder [81], the Ladner-Fischer adder [82], the Ling adder [83], the Brent-Kung adder [84], and the Han-Carlson adder [85]. Another type of adders uses addition blocks of variable sizes, including the carry-select adder [86], the carry-skip adder [87], the conditional-sum adder [88], and the carry-increment adder [89]. The architectures and characteristics of these adders are discussed in [90].

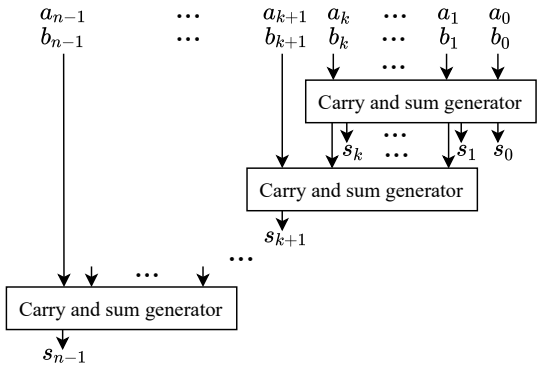


Fig. 1. An approximate speculative adder.

B. Review

Conventional design methodology to accelerate an adder often comes with a cost in circuit area and power dissipation. However, approximate adders trade off accuracy for an overall improvement in hardware efficiency. Based on the approximation schemes to reduce the critical path and hardware complexity, approximate adders are classified into four categories.

1) Speculative adders: As an early scheme, a speculative design leverages the fact that the effective carry chain of an n-bit adder is much shorter than n in most cases [14]. Thus, an n-bit speculative adder uses the previous k bits ($k < n$) to predict the carry for computing each sum bit, as shown in Fig. 1. In this way, the critical path delay is reduced to $O(\log(k))$ (for a parallel implementation such as a CLA, the same below). Compared to the design in [14], the hardware overhead is reduced in the almost correct adder (ACA) by sharing some components among the sub-carry generators [16].

2) Segmented adders: A segmented adder is implemented by several parallel sub-adder blocks with an independent carry-in [17], [19], [91], [92]. Hence, the carry propagation chain is truncated into shorter segments. Fig. 2 shows a basic structure for many segmented adders. As the simplest design, an n-bit equal segmentation adder (ESA) uses $\lceil \frac{n}{k} \rceil$ k-bit sub-adders without any carry-in [19]. Different from ACA, the input bits used for carry computation do not overlap in ESA; thus, for a same k, its hardware cost is significantly lower than ACA.

In an n-bit accuracy-configurable approximate adder (ACAA), $\lceil \frac{n}{k} - 1 \rceil$ $2k$-bit sub-adders are utilized to add $2k$ consecutive bits without carry inputs and k bits are overlapped between two neighboring sub-adders [91]. The accuracy of ACAA can be configured at runtime by changing the bit width of the sub-adders. The generic accuracy configurable adder (GeAr) generalizes the structure of ACAA by varying the number of

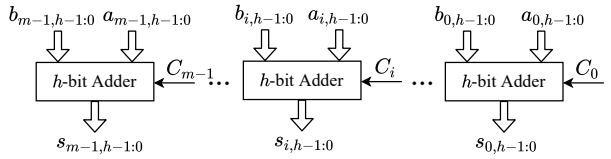


Fig. 2. The basic structure of a segmented adder. $a_{i,h-1:0}$ and $b_{i,h-1:0}$ are the h-bit inputs for the segment i. The inputs can be overlapped between neighboring segments.

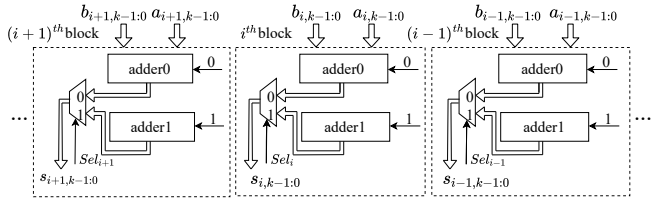


Fig. 3. The approximate carry-select adder with sum selection.

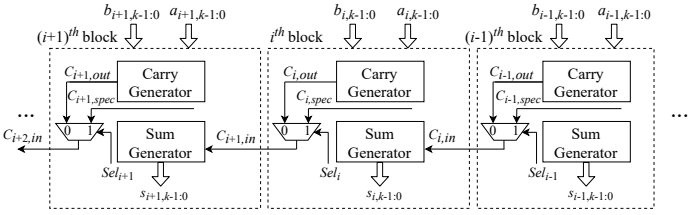


Fig. 4. The approximate carry-select adder with carry-in selection.

overlapped bits used for carry prediction [93], while the quality-area optimal low-latency approximate adder (QuAd) further utilizes sub-adders of variable width [94].

An n-bit error-tolerant adder type II (ETAII) consists of $\lceil \frac{n}{k} \rceil$ k-bit carry generators that are connected in parallel with the k-bit sum generators [17]. For the same k, ETAII utilizes a carry generator to predict the carry for the next sum generator, so it is more accurate than ESA and ACA. However, the circuit of ETAII is more complex than that of ESA, and its delay is larger due to the longer $2k$-bit critical path. For a fixed k, ETAII uses the same carry propagation path as ACAA for each sum, so they share the same error characteristics.

The dithering adder uses a more significant (accurate) sub-adder and a less significant sub-adder with upper and lower bounding modules [68]. An additional control signal is used as the carry-in of the more significant (accurate) sub-adder, which is also used to select the sum output of the less significant sub-adder. To reduce the error due to the ignored carry inputs, an error control and compensation method is developed to trade off computing efficiency for an improved accuracy of a segmented adder in [92].

Generally, the critical paths of the segmented adders are in $O(\log(k))$ due to the carry-ignored segmentation. The circuit complexities are in $O(n\log(k))$ for ESA and ETAII, in $O((n-k)\log(k))$ for ACAA, and in $O\left(\left(\frac{n-L}{k}+1\right)L\log(L)\right)$ for GeAr.

3) Approximate carry-select adders: The structure used in the classic carry-select adder [86] is employed in [95]–[102] to introduce approximation in the selection of the carry-in and sum for each sub-adder. This type of adders is referred to as an approximate carry-select adder with either sum or carry-in selection, as shown in Figs. 3 and 4, respectively. An n-bit approximate carry-select adder consists of $m = \lceil \frac{n}{k} \rceil$ blocks and uses several common signals. For the i^{th} block, generate $g_{i,j} = a_{i,j}b_{i,j}$, propagate $p_{i,j} = a_{i,j} \oplus b_{i,j}$, and $P_i = \prod_{j=0}^{k-1} p_{i,j}$ are defined, where $a_{i,j}$ and $b_{i,j}$ are the j^{th} LSBs of the inputs in block i, where $j = 0, 1, \cdots, k-1$. $P_i = 1$ indicates that all k propagate signals in the i^{th} block are logic "1".

In the speculative carry selection adder (SCSA) [95] and the consistent carry approximate adder (CCA) [99], a sum is selected from adder0 (with carry-in "0") and adder1 (with carry-in "1") by using a multiplexer. In SCSA, the carry-out of adder0 in the $(i-1)^{th}$ block is connected to the Sel_i signal of the multiplexer in the i^{th} block (see Fig. 3). SCSA, ETAII and ACAA achieve the same accuracy for the same value of k due to the identical carry predict function. In CCA, the Sel_i of the multiplexer is determined by the propagate signals in the current and previous blocks. The carry prediction of CCA depends not only on its LSBs, but also on the more significant bits.

In Fig. 4, each block consists of a carry generator and a sum generator. The approximate carry skip adder (CSA) [96], the generate signals-exploited carry speculation adder (GCSA) [100], and the block-based carry speculative approximate adder (BCSA) [103] use different selection schemes for the carry-in of a carry generator. In CSA, the carry-in of the $(i+1)^{th}$ block is determined by the propagate signals of the i^{th} block: it is the carry-out of the $(i-1)^{th}$ sub-carry generator when all propagate signals are true ($P_i = 1$); otherwise it is the carry-out of the i^{th} sub-carry generator. The generate signals are used in GCSA for the carry speculation; the carry-in for the $(i+1)^{th}$ block is selected by its own propagate signals rather than its previous block. The carry-in is the most significant generate signal $g_{i,k-1}$ of the i^{th} block if $P_i = 1$, or else it is the carry-out of the i^{th} carry generator. The carry-in of the $(i+1)^{th}$ block in BCSA is selected between the most significant generate signal $g_{i,k-1}$ and the carry-out of the i^{th} block, that is, $C_{i+1,in} = \overline{S_i}C_{i,out} + S_i g_{i,k-1}$, where $S_i = \overline{a_{i+1,0} + b_{i+1,0}} + g_{i,k-1}$ [103]. An error detection and recovery scheme is further proposed to partially compensate the errors by modifying the LSB of the sum output in each block.

In the carry speculative adder (CSPA), each block contains one sum generator, two internal carry generators with carry-0 and carry-1, respectively, and a simple carry predictor [98]. The carry-out of the i^{th} carry predictor selects a carry-in for the $(i+1)^{th}$ sum generator. The carry predictor uses k_l rather than k input bits ($k_l < k$), so it leads to a simpler circuit than SCSA for the same block size k.

Some control signals are added to the gracefully-degrading accuracy-configurable adder (GDA) to configure the accuracy by selecting an accurate or approximate carry-in for each sub-adder [97]. Thus, the delay of GDA varies with the carry propagation path determined by the control signals.

In the carry cut-back adder (CCBA), the full carry propagation is prevented by a multiplexer or an OR gate [102]. The carry-in for a segment is determined by a cut signal from a carry propagate block at a higher position, a carry speculated from a short chain at a lower position, and the carry-out of the previous segment. The delay and accuracy of CCBA depend on the distance between the propagate block and the multiplexer or OR gate.

The critical path delay of the approximate carry-select adders can be given by $O(\log(k))$, when the bit width of the input

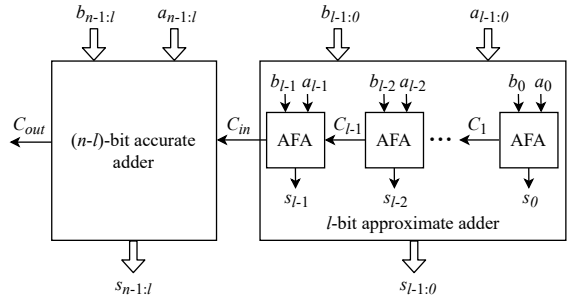


Fig. 5. An n-bit adder using approximate full adders (AFAs).

operands in each block is k. The circuit area varies with the complexity of the carry prediction and selection schemes.

4) Approximate full adders: Another method for reducing the critical path and power dissipation of an adder is to approximate a full adder. The approximate full adder (AFA) is then used to implement l LSBs in an n-bit adder ($l < n$), while the $(n-l)$ MSBs are computed by an accurate adder, as shown in Fig. 5. In the lower-part-OR adder (LOA), an OR gate is used as a simple AFA, and one AND gate is used to generate the carry-in for the accurate adder [18].

Other AFA designs include the mirror adder [20], the approximate XOR/XNOR-based full adders [104], the inexact adder cells proposed in [105], and the approximate reverse carry propagate full-adder [106] (specific for the RCA structure). Additionally, emerging technologies such as magnetic tunnel junctions have been considered for the design of AFAs for a shorter delay, a smaller area and a lower power consumption [107], [108]. Finally, a simply truncated adder (TruA) that works with a lower precision is considered as a baseline design.

The critical path of this type of adders is typically in $O(\log(n-l))$ when there is no carry propagation for the AFAs. LOA is selected as the reference design in the evaluation due to its logic-level implementation while most other AFAs are designed at the transistor level.

In addition to the above four categories, a library of 430 approximate 8-bit adders has been automatically generated by using Cartesian genetic programming (CGP) and a multi-objective genetic algorithm [109]. Due to the restricted bit width of 8 bits, however, this group of designs is not considered in the evaluation.

C. Evaluation

In this evaluation, we consider 16-bit approximate adders. In the circuit implementations, all sub-adders in the designs are implemented by CLAs for a high efficiency.

To obtain the error characteristics, the functions of the 16-bit approximate adders are implemented in MATLAB and simulated with 10 million uniformly distributed random input combinations. The simulation results show similar trends in MRED and NMED for the approximate adders [51], so only the MRED is reported here. For circuit measurements, the 16-bit approximate adders are implemented in HDLs and synthesized as described in Section II-B2. The clock period used in the power estimation is 1 ns.

Fig. 6 shows the delay comparison of approximate adders with respect to MRED and ER (for delay-optimized synthesis), while Fig. 7 shows the power comparison (for area-optimized synthesis). The overall comparison in PDP and MRED is shown in Fig. 8 (for both delay- and area-optimized syntheses). A Pareto front is delineated in each figure to show the designs with the highest efficiency. As ETAII, ACAA and SCSA share the same error characteristics for a certain k, only ETAII is shown in the figures due to its lower hardware overhead.

TABLE I. Summary of approximate adders.

Adder	Error characteristics		Circuit measurements		
	ER	ED	Performance (delay-optimized)	Power (area-optimized)	PDP
ESA	high	high	high	low	low
GeAr	low		high		
CSA	low	low			
CSPA		high	high		
CCBA	high	low		low	low
LOA	high	low		low	low
TruA	high			low	low

As can be seen in Fig. 6, most approximate adders have very close ERs between 0.5% to 35% except for GeAr (R4_P8) and CSA (for $k > 3$) with ERs smaller than 0.5%, and CCBA, ESA, LOA and TruA with very high ERs, although CCBA, LOA and TruA can have relatively small MREDs. CSA can be very accurate, whereas ESA, BCSA and CSPA show a low accuracy with relatively large MREDs and ERs.

Performance/Power vs. accuracy: As also shown in Fig. 6, LOA, CCBA and GeAr can be faster than the other designs for a relatively small MRED, so they exhibit a balanced tradeoff in performance and accuracy (in MRED). ESA and CSPA are the fastest at a large MRED and ER. For a similar ER, some configurations of CSA, GeAr and ETAII can be faster than others (i.e., in the Pareto front). As shown in Fig. 7, CCBA, LOA and TruA achieve the best power and accuracy tradeoffs (in terms of MRED). GeAr, BCSA and CCBA are in the Pareto set for power consumption and ER.

Energy vs. accuracy: To consider both error and circuit characteristics, the MRED and PDP are selected as representative metrics. As shown in Fig. 8, a similar trend is obtained for both the delay- and area-optimized syntheses in the PDP and MRED of the approximate adders. Overall, CCBA, LOA and TruA achieve the best tradeoffs between accuracy (in MRED) and energy (in PDP); however, they have the highest ERs. Nevertheless, these approximate adders show a decent tradeoff in error magnitude and hardware efficiency. In particular, they are suitable for applications in which a high ER is not as detrimental as a large error magnitude.

In summary, truncation is an effective approach to a hardware-efficient design, albeit resulting in a high ER. On the other hand, the carry select scheme can be very effective in highly accurate designs such as the CSA. A speculative adder results in a very high power dissipation and a large error magnitude (e.g., ACA). The advantages and disadvantages of the approximate adders with at least one prominent property are summarized in Table I. In this table, ED stands for both MRED and NMED. As can be seen, the ESA is very hardware-efficient for applications with high error tolerance, whereas CSA is suited for applications that require a high accuracy. When a high ER is not an issue, CCBA, LOA and TruA are the most-efficient designs.

IV. Approximate Multipliers

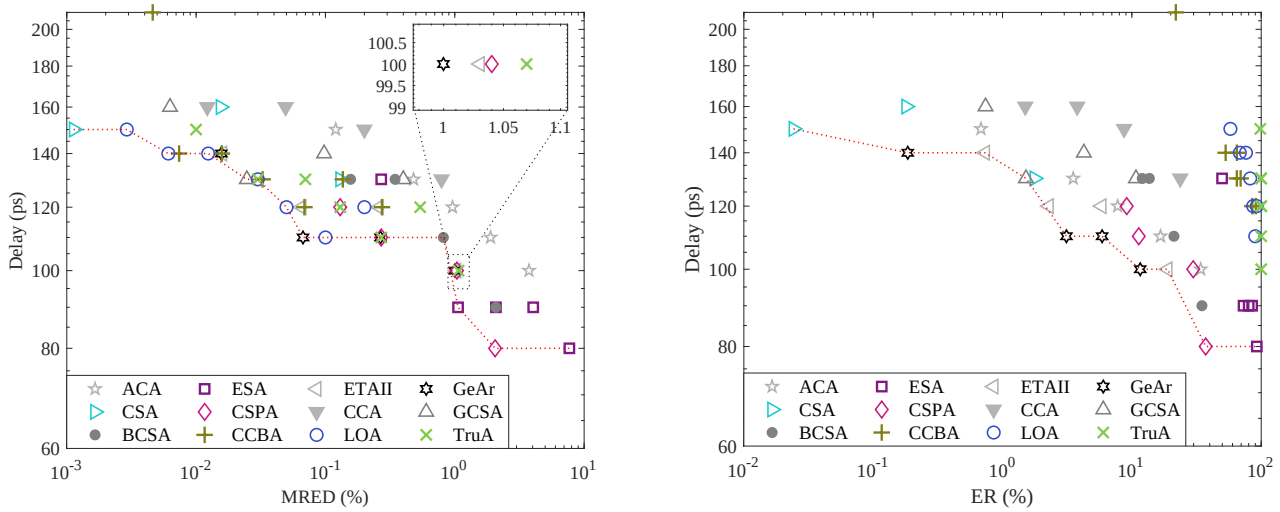
A. Preliminaries

Typically, a combinational multiplier consists of three processing stages, partial product (PP) generation, PP accumulation and

a final carry propagate addition, as shown in Fig. 9. Let the two input operands of an $n \times n$ unsigned multiplier be $A = \sum_{i=0}^{n-1} A_i 2^i$ and $B = \sum_{i=0}^{n-1} B_i 2^i$, where A_i and B_i are the i^{th} least significant bits of inputs A and B, respectively, and i starts from 0. A partial product is often generated by an AND gate, i.e., $PP_{i,j} = A_j B_i$. To accumulate the PPs, three structures are widely used: the carry-save adder array [110], the Wallace tree [111], and the Dadda tree [112].

Fig. 10 shows a carry-save adder array for a 4×4 unsigned multiplier, where the carry and sum signals generated by the adders in a row are passed on to the adders in the next row. The carry signals propagate through the adders in a diagonal direction. Hence, the critical path for an $n \times n$ multiplier is approximately in $O(n)$. Due to its regular layout, the array structure in Fig. 10 requires mostly short wires and is easy to scale to large arrays.

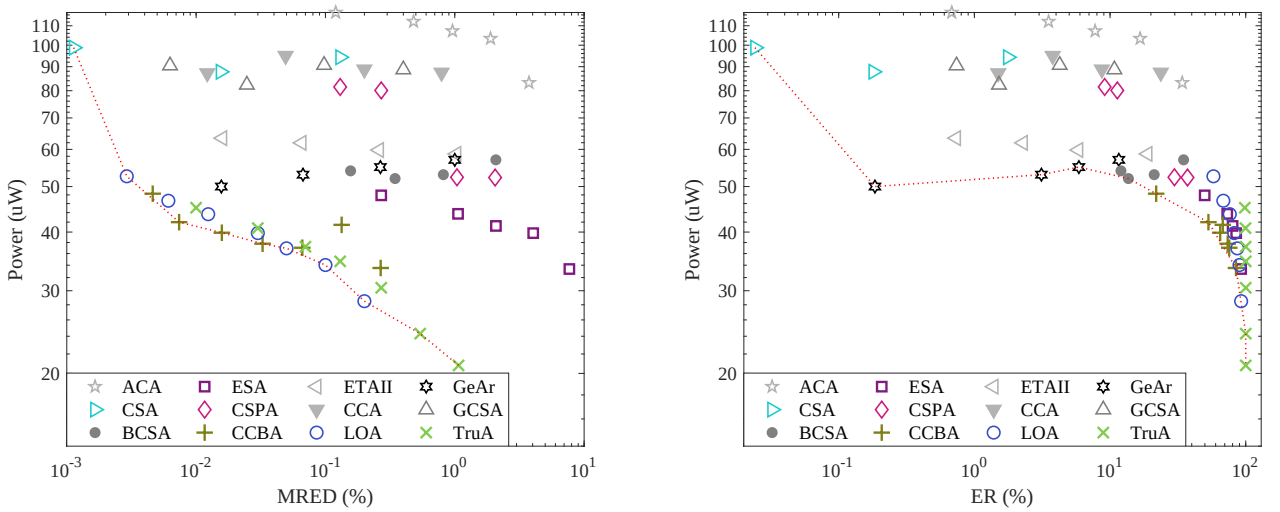
A Wallace tree utilizes FAs, half adders (HAs) and 4:2 compressors for a fast accumulation of the PPs, as shown in the dotted box in Fig. 9 for a 4×4 unsigned multiplier. The adders in each stage operate in parallel without carry propagation, and the same operation repeats until two rows of the PPs are left. For an $n \times n$ multiplier, about $\lceil \log_{1.5}(n/2) \rceil$ stages are required in a Wallace tree [110]. Therefore, the delay is in $O(\log(n))$, which is shorter than that of the array structure. The Dadda tree has a similar structure as the Wallace tree, but it uses as few adders as possible rather than reducing PPs as early as possible in a



(a) Optimized delay vs. MRED

(b) Optimized delay vs. ER

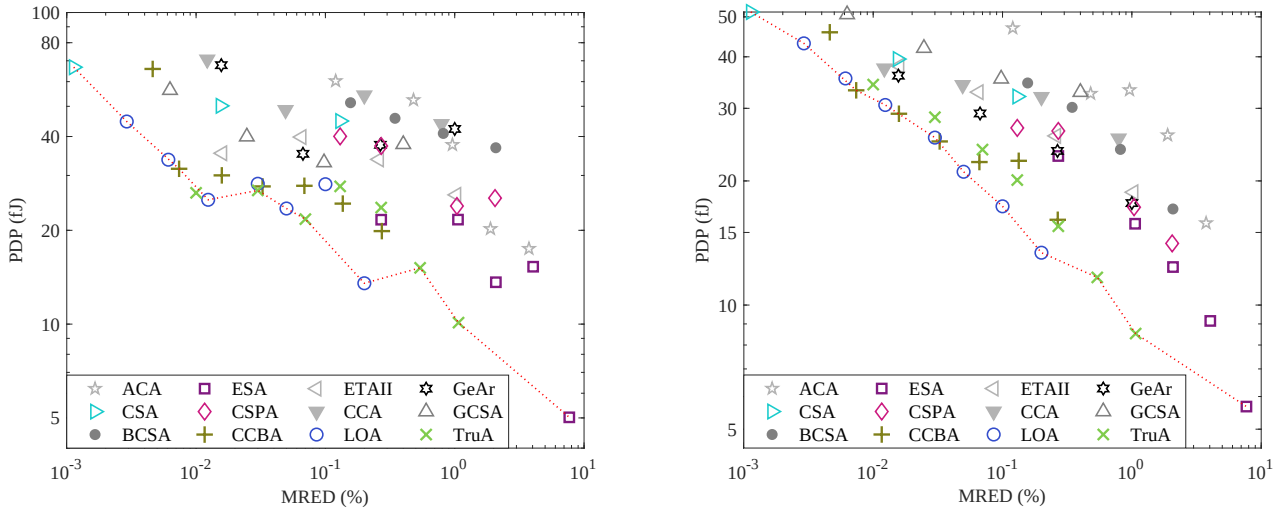
Fig. 6. Optimized delay vs. accuracy for the approximate 16-bit adders using different error metrics. *Note*: The number of approximate or truncated LSBs for LOA and TruA ranges from 3 to 9, the sub-adder width of ESA is from 8 down to 3, the number of bits used for carry speculation for ACA is from 8 down to 3 from left to right. The block width for CSA is from 5 down to 3, and it is from 6 down to 3 for the other adders from left to right. In CSPA, the size of the carry predictor is $\lceil k/2 \rceil$. The global speculative carry for CCA is "0," which leads to a more accurate result than using "1." For GeAr, the configurations from left to right are R4_P8, R6_P4, R4_P4, and R2_P4 (Rm_Pk means k previous bits are used for generating m bits of sum results). For CCBA, the configurations with the smallest PDPs are chosen for a similar MRED.



(a) Power (area-optimized) vs. MRED

(b) Power (area-optimized) vs. ER

Fig. 7. Power consumption vs. accuracy for the area-optimized approximate 16-bit adders using different error metrics.



(a) PDP (delay-optimized) vs. MRED

(b) PDP (area-optimized) vs. MRED

Fig. 8. A comparison of power-delay product (PDP) and MRED for the approximate 16-bit adders.

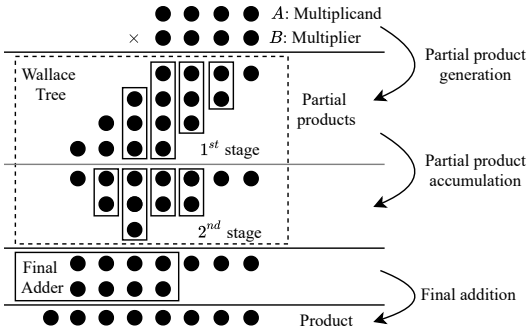


Fig. 9. The basic arithmetic process of a 4×4 unsigned multiplication. ●: an input, a partial product or an output bit; □ : a half adder, a full adder or a 4:2 compressor.

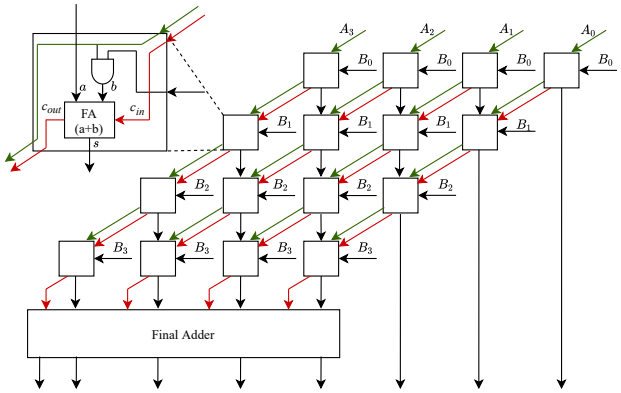


Fig. 10. A 4×4 unsigned multiplier using a carry-save adder array.

Wallace tree. Compared to the array structure, a tree-based PP accumulation is faster; however, a tree structure requires longer and more complex wiring, which can result in a larger circuit area [113].

Signed multiplication uses 2's complement representation. The input operands are given as $A=-A_{n-1}2^{n-1}+\sum_{i=0}^{n-2}A_i2^i$ and $B=-B_{n-1}2^{n-1}+\sum_{i=0}^{n-2}B_i2^i$. The Booth algorithm is then used to recode the multiplier for generating PPs [114]. MacSorley modified the Booth algorithm to the radix-4 Booth algorithm [115], which reduces the number of PPs by half. The radix-2^r Booth algorithm can be obtained by using the same principles of the radix-4 scheme. In addition, the Baugh-Wooley algorithm [116] and the modified Baugh-Wooley algorithm [117] can simplify the signed multiplication by adding the 2's complements of the PP rows and preprocessing the constant additions. The modified Baugh-Wooley algorithm is also widely used to eliminate the sign extension in Booth multipliers [118].

To approximate an unsigned multiplier, five methodologies have been considered: 1) approximation in generating the PPs [22], 2) approximation (including truncation) in the PP tree [18], [21], [119], [120], 3) using approximate adders [121], counters [63] or compressors [79], [80], [122]–[126] in the PP reduction, 4) using logarithmic approximation [11], [64], [127]–[130], and 5) using an automated process such as a genetic programming method [109], [131]. For signed multiplication, approximate Booth multipliers have been designed for its fast operation on a reduced number of PPs.

Therefore, approximate multipliers are classified into five unsigned categories and signed Booth multipliers.

B. Approximate Unsigned Multipliers

1) Approximation in generating partial products: As an early design, the underdesigned multiplier (UDM) utilizes an approximate 2×2 multiplier to construct larger multipliers [22]. The 2×2 multiplier approximates the product "1001" with "111" when both the inputs are "11," so saving one output bit and simplifying the logic circuit. The ER of this 2×2 multiplier is $0.5^4=6.25\%$ if each input bit is equally likely to be "0" or "1".

2) Approximation in the partial product tree: A broken-array multiplier (BAM) omits some carry-save adders in an array multiplier in both the horizontal and vertical directions [18]. A more straightforward approximation is to truncate some LSBs on the input operands so that a smaller multiplier is sufficient for the remaining MSBs. This truncated multiplier (TruM) is considered as a baseline design for comparison. Different from BAM and TruM, several consecutive rows of PPs that do not necessarily

start from the LSB are ignored for the PP reduction in [132]. This design is referred to as a partial product perforation-based multiplier (PPAM).

The error tolerant multiplier (ETM) consists of a multiplication section, a non-multiplication section, and a control block [21]. The NOR gates-based control block determines: i) if all k MSBs in at least one of the two n-bit input operands are zeros, the multiplication section (using an accurate $k \times k$ multiplier, where $k < n$) is activated to multiply the LSBs without any approximation, and ii) otherwise, the accurate multiplier is used to multiply the MSBs, while the non-multiplication section is used to approximately process the LSBs. The static segment multiplier (SSM) uses a similar partition scheme, but the approximation section is omitted for not processing the LSBs [133]. If the MSBs of one input are all zeros, its LSBs are multiplied by either the MSBs or the LSBs of the other input depending on whether MSBs are all zeros.

Similarly, an exact $k \times k$ sub-multiplier is used in the design of an $n \times n$ dynamic range unbiased multiplier (DRUM) [120]. However, the k-bit inputs of the reduced-width multiplier are dynamically selected starting from the leading "1"s or the most significant "1"s in the two n-bit input operands. If the leading "1" position is higher than k, the redundant LSBs are truncated and the LSB of the k selected bits is set to "1". Otherwise, the leading "1" position is ignored, and the k LSBs of the input operands are selected as the inputs of the sub-multiplier. The final output is then obtained by using a barrel shifter to restore the computed result. As the input bits are more effectively processed, DRUM is more accurate than ETM and SSM. Moreover, it produces unbiased errors, so it is suited for accumulative operations. However, it uses a more complex circuit for the dynamic selection of inputs.

An approximate Wallace tree multiplier (AWTM) utilizes a bit-width aware approximate multiplication and a carry-in prediction [119]. An $n \times n$ AWTM is implemented by four $n/2 \times n/2$ sub-multipliers, where the most significant sub-multiplier $A_H B_H$ is further divided into four $n/4 \times n/4$ sub-multipliers. By using different numbers of approximate $n/4 \times n/4$ sub-multipliers in $A_H B_H$, the AWTM is configured into four modes. The three less significant $n/2 \times n/2$ sub-multipliers ($A_H B_L$, $A_L B_H$ and $A_L B_L$) are approximate.

3) Using approximate counters or compressors in the partial product reduction: An inaccurate 4×4 Wallace multiplier uses a 4:2 counter that approximates the output of the carry and sum, "100," with "10" when all four inputs are "1" [63]. For uniformly distributed inputs, the probability of one bit being "1" is 0.5, so the probability that a partial product is "1" is 0.25. The error rate of the approximate 4:2 counter is, therefore, only $0.25^4 = 0.39\%$. Larger multipliers can be constructed using the inaccurate counter-based 4×4 multiplier (ICM, in general). In the approximate counters in [134], the more significant output bits are ignored for an efficient implementation of several signed multipliers.

Two approximate designs that implement simplified functions of 4:2 compressors are considered for Dadda multipliers [123]. To lower the error probability, the PPs are encoded by propagate (i.e., $PP_{i,j} + PP_{j,i}$) and generate (i.e., $PP_{i,j} \cdot PP_{j,i}$) signals, which enable the design of several approximate compressors with a relatively low error probability [124]. Similarly, an approximate 4:2 compressor with encoded inputs is proposed for 4×4 multipliers, which is then used to construct larger multipliers [125]. The dual-quality 4:2 compressors in [79] can switch between exact and approximate operating modes using power gating techniques. These compressors are then used in the PP accumulation of a Dadda multiplier and the accuracy can be dynamically configured. Using a 3-input majority gate, a FinFET-based imprecise 4:2 compressor is designed for an approximate 8×8 Dadda multiplier with truncation in the PP array [80].

In the high-order compressor based multiplier (HOCM), each column of PPs is accumulated by only one compressor [126]. An allocation algorithm is then developed to determine the use of exact and approximate compressors at different stages of the accumulation with the truncation of the lower half PPs.

In [121], a novel approximate adder uses two adjacent inputs to generate a sum and an error bit for accumulating the PPs. To alleviate the error due to the approximate adder, two error recovery schemes are considered to use either OR gates to accumulate the error bits in the so-called approximate multiplier 1 (AM1), or both OR gates and the approximate adders in the approximate multiplier 2 (AM2). Moreover, TAM1 and TAM2 are obtained by truncating the lower half of the PPs in AM1 and AM2, respectively [50], [135].

4) Using logarithmic approximation: Mitchell's algorithm leverages the logarithmic and anti-logarithmic approximations of a binary number. It serves as the basis of logarithmic multipliers (LMs) [11]. In this algorithm, the two unsigned binary input operands A and B of a multiplier are expressed as

$$A = 2^{k_1}(1 + x_1) \tag{5}$$

and

$$B = 2^{k_2}(1 + x_2), \tag{6}$$

where k_1 and k_2 indicate the leading "1" positions of A and B, respectively; x_1 and x_2 are the fractional numbers that represent the bits to the right of the leading "1"s normalized by 2^{k_1} and 2^{k_2}, respectively. The product of A and B is then given by

$$M = A \times B = 2^{k_1+k_2}(1 + x_1)(1 + x_2). \tag{7}$$

Thus,

$$\log_2 M = k_1 + k_2 + \log_2(1 + x_1) + \log_2(1 + x_2). \tag{8}$$

As $0 \leq x_1, x_2 < 1$, $\log_2(1 + x_1) \approx x_1$ and $\log_2(1 + x_2) \approx x_2$. Hence, (8) is approximated by

$$\log_2 M \approx k_1 + k_2 + x_1 + x_2. \tag{9}$$

The multiplication is then completed by performing an antilogarithmic approximation, i.e.,

$$M \approx \begin{cases} 2^{k_1+k_2}(x_1 + x_2 + 1) & \text{if } x_1 + x_2 < 1 \\ 2^{k_1+k_2+1}(x_1 + x_2) & \text{if } x_1 + x_2 \geq 1 \end{cases} \tag{10}$$

To improve the accuracy of an LM, the ALM-SOA uses a truncated binary-logarithm converter and a set-one-adder (SOA) for the addition [127]. The SOA simply sets the LSBs to constant "1"s, and generates a carry-in for the MSBs using an AND gate. Moreover, an improved algorithm using exact and approximate adders (ILM-EA and ILM-AA) is proposed in [64], [128]. In [129], the input operands between two consecutive powers of two are partitioned into several segments. An error reduction factor is

then analytically determined for each segment and compensated to the result of the basic LM. A two-stage design that uses two truncated LMs for error correction achieves a low and unbiased average error [130].

5) Using an automated process: In [109], 471 8×8 approximate unsigned multipliers are automatically generated by using CGP and a multi-objective genetic algorithm. As CGP can provide better implementations of a circuit than conventional synthesis tools, it is used to denote a circuit using this design method. An approximate circuit is generated by randomly removing some connections of several accurate designs. A genetic algorithm is then applied for design space exploration to obtain the optimal approximate circuits with respect to MRED. These 8×8 multipliers are then used to construct 16×16 approximate multipliers, referred to as CGPM1 to CGPM6 [131].

C. Evaluation of Approximate Unsigned Multipliers

The error and circuit characteristics of 16×16 approximate unsigned multipliers are obtained with the same experimental setup as in the evaluation of approximate adders, except that the clock period used in the power estimation is 4 ns.

As shown in [51], most of the approximate multipliers result in large ERs close to 100%. However, ICM has a low ER of 5.45% because only one approximate counter with an ER of 0.39% is used in a 4×4 multiplier block. Some configurations of CGPM1, CGPM2 and CGPM3 also show lower ERs than the other designs. Additionally, the ER of UDM is 80.99%, lower than most of the other designs. Hence, the accuracy is only compared here in MRED and NMED. For circuit measurements, Fig. 11 shows the critical path delay of the multipliers synthesized under delay-optimized constraints with respect to MRED and NMED, while Fig. 12 shows the power consumption for area-optimized synthesis. Based on the preliminary results in [57], the designs with good tradeoffs are selected from each category for comparison here. The array and Wallace architectures are considered for TruM, which are denoted as TruMA and TruMW, respectively.

Performance vs. accuracy: Fig. 11 shows that most approximate unsigned multipliers exhibit a similar performance trend vs. both MRED and NMED except for ICM and DRUM. CGPM1 is the most accurate design with very small values of MRED and NMED, and a reasonable performance. As expected, the LMs (ALM-SOA and ILM-AA) are good in performance, but poor in accuracy. HOCM, TAM1, ILM-AA and ALM-SOA show the best tradeoffs between performance and accuracy. PPAM can have the shortest delay but largest error.

Power vs. accuracy: As shown in Fig. 12, LMs are very power-efficient albeit with a very low accuracy, whereas CGPM1 is relatively power-hungry but with a high accuracy. At a medium accuracy, BAM consistently consumes a low power, followed by CGPM3. Some configurations of HOCM also show good tradeoffs in power-efficiency and accuracy. HOCM (1StepTrunc) shows the best tradeoff in power and accuracy.

Energy vs. accuracy: The PDPs of the unsigned multipliers with respect to MRED are shown in Fig. 13. The overall trend is slightly different between the delay-optimized (Fig. 13(a)) and area-optimized (Fig. 13(b)) synthesis results. As shown in Fig. 13(a), the 1StepFull in HOCM, TAM1, CGPM1 and CGPM3 exhibit the best energy-accuracy tradeoffs, located in the center of the plot. In Fig. 13(b), CGPM3, HOCM, TAM1 and TruMA show slightly better tradeoffs than the other designs. At a very low accuracy, ALM-SOA and PPAM are the most hardware-efficient with the smallest PDP values.

TABLE II. Summary of approximate unsigned multipliers.

Multiplier	Error characteristics		Circuit measurements		
	MRED	NMED	Performance (delay-optimized)	Power (area-optimized)	PDP
BAM			low	low	low
HOCM			high		
TAM1			high		low
CGPM1	low	low		high	
ALM-SOA	high	high	very high	very low	very low
ILM-AA	high	high	very high	low	low
TruMA				low	low
TruMW			high	low	low

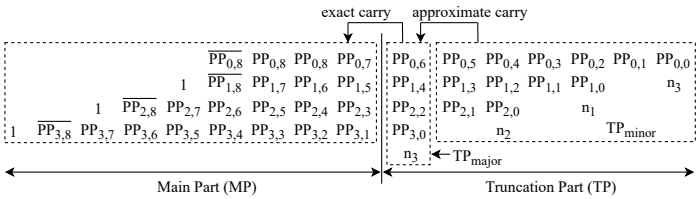


Fig. 14. The partial product (PP) partition for an 8×8 fixed-width modified Booth multiplier [136]. $PP_{i,j}$ is the j^{th} PP in the i^{th} PP vector, $\overline{PP_{i,j}}$ is the inverted $PP_{i,j}$, and n_i is the sign of the i^{th} encoded digit.

In summary, truncation is effective in reducing the delay and energy consumption of unsigned multipliers. An LM tends to be hardware-efficient but with a rather low accuracy. The automatically-generated multipliers can be highly accurate with a reasonable hardware consumption. A brief summary of the error and circuit characteristics is shown in Table II for the approximate unsigned multiplier in the Pareto fronts.

D. Approximate Booth Multipliers

The modified (or radix-4) Booth algorithm is commonly used in the design of approximate Booth multipliers [118], [136]–[140]. Initially aimed at a fixed-width signed multiplier, a widely used method is to truncate the lower half of the PPs to generate an output with the same width as the input. This truncation saves the circuits for PP accumulation, but it introduces a large error. Hence, many error compensation schemes have been proposed to increase accuracy [118], [136], [137], [139].

Inspired by the BAM, the broken Booth multiplier (BBM) omits the adder cells to the right of a vertical line [138], whereas directly truncating k LSBs of the input operands leads to a truncated Booth multiplier (TBM-k). The TBM is considered as a baseline design for comparing the Booth multipliers.

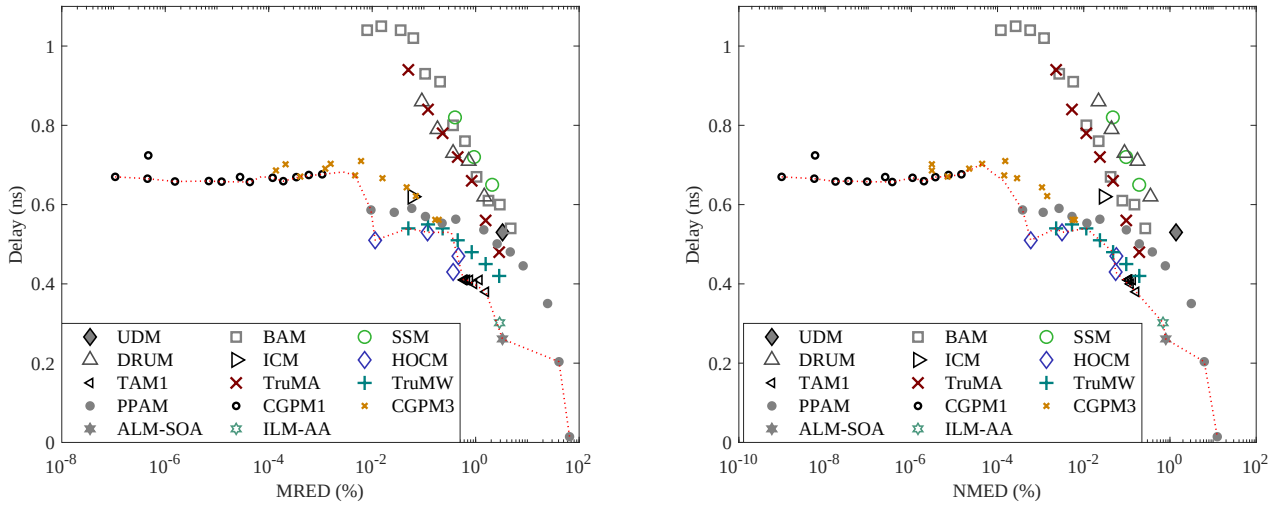
Generally, a fixed-width Booth multiplier is based on a partition of the PP array, as shown in Fig. 14. For an 8×8 fixed-width modified Booth multiplier, the PP array is divided into two parts, the higher half denoted as the main part (MP) and the lower half truncation part (TP). The TP is further divided into TP_{major} and TP_{minor}. The final product of a fixed-width multiplier is the addition of the MP and the carry signals generated from the TP.

In [136], the carry signals include the exact carry from the TP_{major} and an approximate carry from the TP_{minor} (Fig. 14). The approximate carry is generated by the output of the modified Booth encoders. This multiplier is referred to as BM04. Using a similar partition scheme, BM11 relies on a simplified sorting

network to generate the carries for error compensation [118]. This error compensation makes the errors symmetrical and centred around zero, which reduces the error bias and mean-squared error. In BM15, the error due to truncation is compensated by the outputs of the Booth encoders and the multiplicand [141]. In BM07, the number of PP columns in TP_{major} is adaptively variable to compensate for the quantization error in a fixed-width multiplier [137]. Another design is based on a probabilistic estimation-based bias [139], referred to as PEBM. In this design, an error compensation formula is derived from a probability analysis, where the number of PP columns in TP_{major} varies in accordance with the desired trade-off between hardware and accuracy.

To reduce the additional delay due to the radix-8 Booth algorithm, an approximate recoding adder is proposed for calculating the triple multiplicands in [142]. A Wallace tree and a truncation technique are then utilized for the PP accumulation. To be consistent with the fixed-width Booth multipliers, the most efficient approximate radix-8 Booth multiplier, ABM2_R15 (with the truncation of 15 bits, resulting in a fixed-width multiplier), is considered and denoted as ABM2.

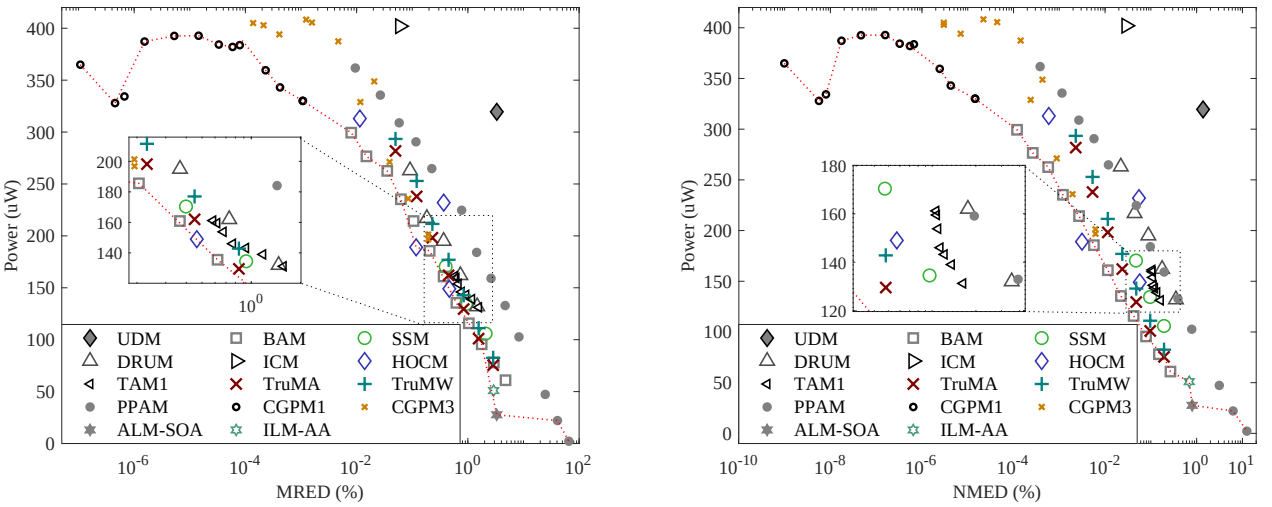
To speedup the PP generation, two approximate radix-4 Booth encoders are proposed by simplifying the K-Map to generate k least significant PP columns for an $n \times n$ multiplier ($k = 1, 2, \cdots, 2n$) [140]. By changing the value of k, different tradeoffs can be achieved between accuracy and hardware efficiency.



(a) Optimized delay vs. MRED

(b) Optimized delay vs. NMED

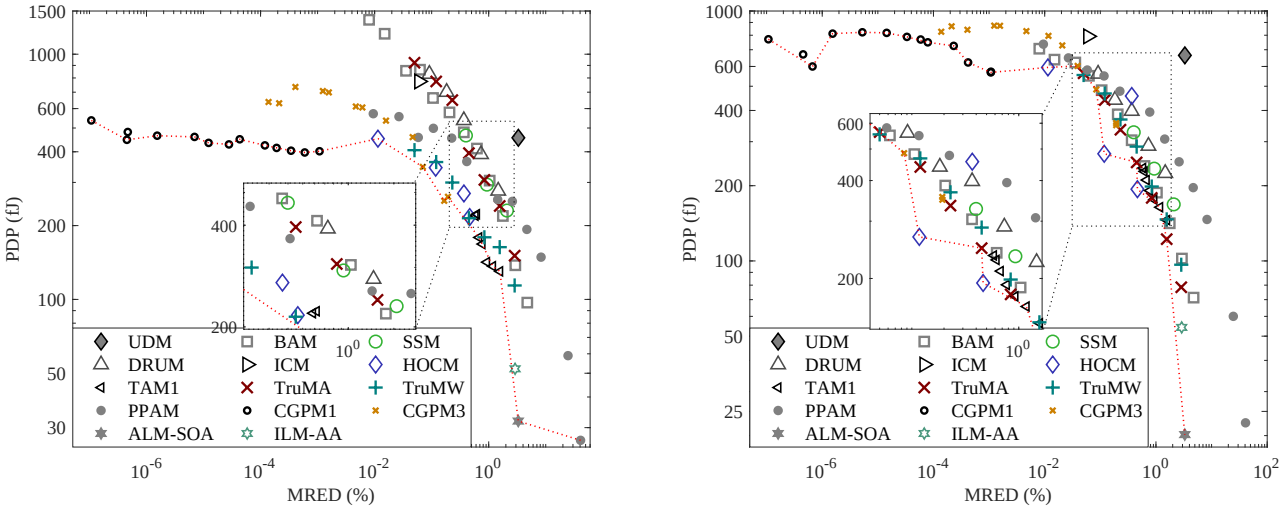
Fig. 11. Optimized delay vs. accuracy for the approximate 16×16 unsigned multipliers using different error metrics.
Note: The number of truncated LSBs for TruMA and TruMW is from 2 to 8 from left to right, and from 11 to 22 for BAM. The number of MSBs used for error compensation is from 16 to 10 for TAM1. The size of the accurate sub-multiplier is from 10 to 8 for SSM, and 10 to 6 for DRUM. The configurations for HOCM are 1StepFull (with approximate compressors in the first accumulation stage), 1StepTrunc (1StepFull with a truncation of LSBs), 2StepFull (with approximate compressors in both the first and second stages), 2StepTrunc (2StepFull with a truncation of LSBs) from left to right. For CGPM1 and CGPM3, which respectively use one and three 8×8 approximate multipliers for constructing a 16×16 multiplier, the configurations with the smallest PDPs are shown for a specific MRED, selected from 500 configurations for each design.



(a) Power (area-optimized) vs. MRED

(b) Power (area-optimized) vs. NMED

Fig. 12. Power consumption vs. accuracy in MRED and NMED for the area-optimized approximate 16×16 unsigned multipliers.



(a) PDP (delay-optimized) vs. MRED

(b) PDP (area-optimized) vs. MRED

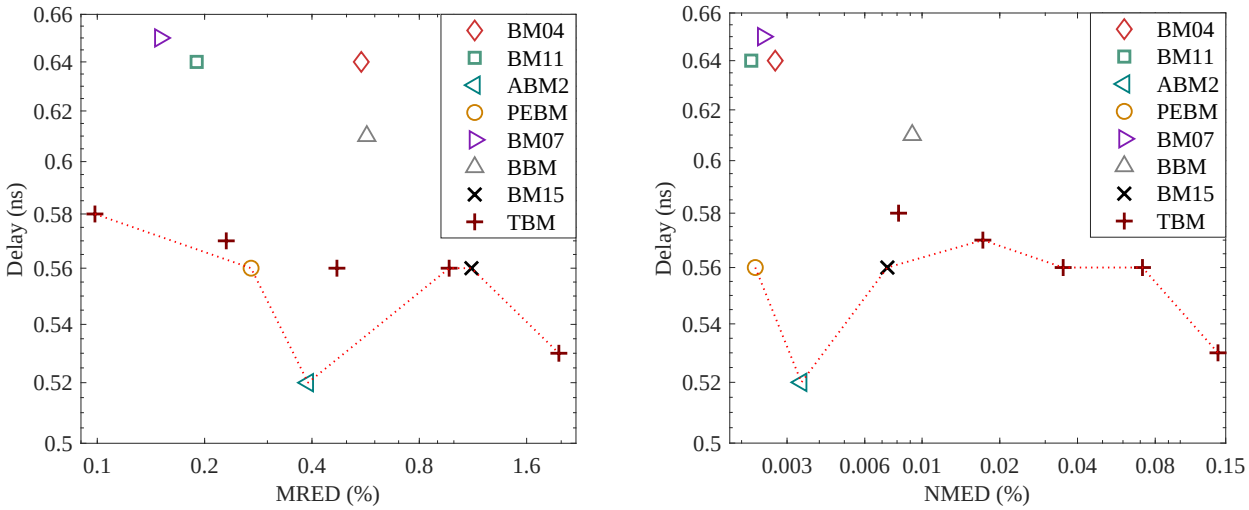
Fig. 13. A comparison of power-delay product and MRED for the approximate 16×16 unsigned multipliers.

E. Evaluation of Approximate Booth Multipliers

In this evaluation, we consider 16×16 approximate (or fixed-width) Booth multipliers for signed multiplication. The clock period for the power estimation is 4 ns. Fig. 15 shows the optimized delay with respect to NMED and MRED, while the power is shown in Fig. 16 for area-optimized synthesis. Fig. 17 shows the tradeoff between PDP (for both delay- and area-optimized syntheses) and MRED for the approximate Booth multipliers.

Performance/Power vs. accuracy: As revealed in Figs. 15 and 16, most fixed-width Booth multipliers show similar NMEDs except for BBM and BM15 with relatively large values. Compared to the fixed-width Booth multipliers, TBM can have a similar MRED and higher NMED, with a higher speed and power dissipation. With a moderate accuracy, ABM2 is the fastest and the most power-efficient. With a very high accuracy, BM07 is the slowest design with a relatively high power consumption, followed by BM11. PEBM shows a moderate speed and power dissipation, with a relatively high accuracy.

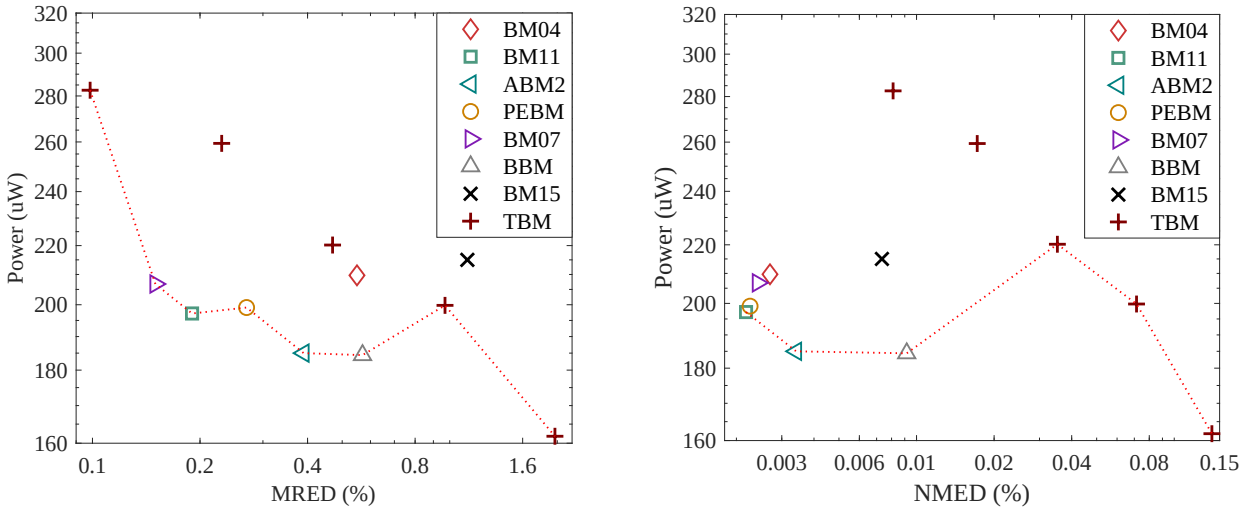
Energy vs. accuracy: Fig. 17 shows that BM07, BM11 and PEBM exhibit the best tradeoffs between accuracy and PDP. ABM2 and BBM stand out too for power-optimized synthesis. A summary of the error and circuit characteristics is shown in Table III. Overall, BM07 and BM11 are relatively accurate but slow. PEBM shows small values of NMED and PDP, as well as a high speed. ABM2 is efficient in both power and performance with a moderate accuracy.



(a) Optimized delay vs. MRED

(b) Optimized delay vs. NMED

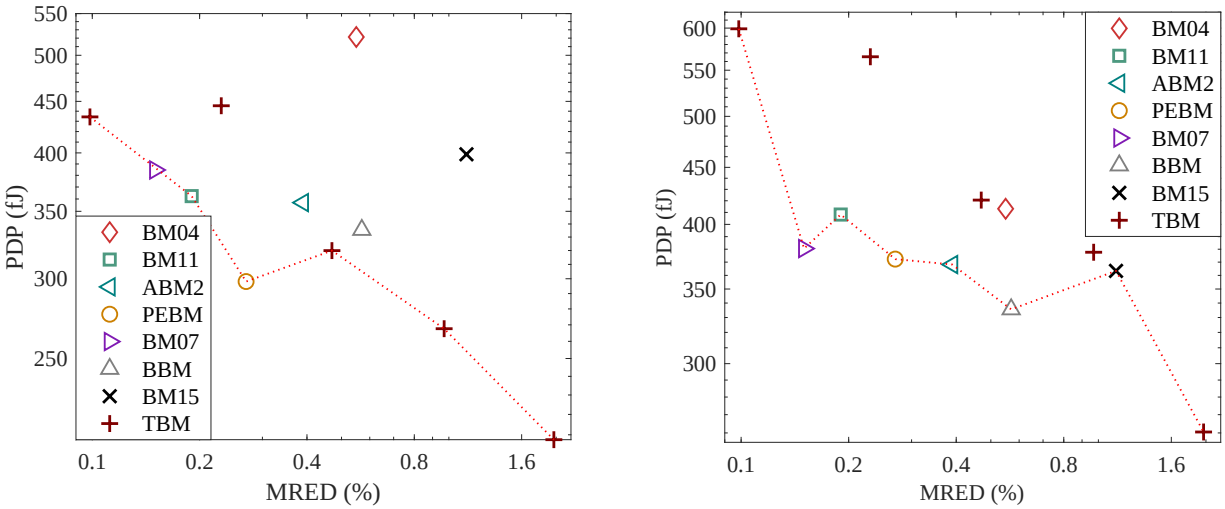
Fig. 15. Delay vs. accuracy for the approximate 16×16 Booth multipliers. The number of truncated LSBs for TBM is from 2 to 6 from left to right.



(a) Power (area-optimized) vs. MRED

(b) Power (area-optimized) vs. NMED

Fig. 16. Power consumption vs. accuracy for the approximate 16×16 Booth multipliers.



(a) PDP (delay-optimized) vs. MRED

(b) PDP (area-optimized) vs. MRED

Fig. 17. A comparison of power-delay product and MRED for the approximate 16×16 Booth multipliers.

TABLE III. Summary of the approximate Booth multipliers.

Multiplier	Error characteristics		Circuit measurements		
	MRED	NMED	Performance (delay-optimized)	Power (area-optimized)	PDP
BM07	low	low	low		
BM11	low	low	low		
PEBM		low	high		low
BBM	high			low	low
ABM2			high	low	

V. Approximate Dividers

A. Preliminaries

Although the divider is not as frequently used as adders and multipliers, the system performance can be significantly degraded if it is not appropriately implemented, whereas it is hard to reduce the latency of dividers without significant overhead in area [143]. A straightforward approach to divider design is to follow the pencil-and-paper algorithm. In general, the quotient of a division is computed by iteratively subtracting a multiple of the divisor from the partial remainder that is initially set to the dividend. In a restoring divider, the partial remainder is corrected or reserved when the subtraction yields a negative number, while it is not corrected in a nonrestoring divider [110]. Fig. 18 shows an $8/4$ unsigned restoring array divider that uses a multiplexer and the borrow signal in the subtractor cell to retain the partial remainder. Generally, n^2 subtractor cells are required in a $2n/n$ array divider. The critical path is in $O(n^2)$ due to the ripple borrow propagations among subtractor cells, whereas it is in $O(n)$ for an $n \times n$ array multiplier. Therefore, an array divider is much slower than an array multiplier. However, the delay of an array divider can be reduced by using carry-save reduction and carry-lookahead principles, with an increased cost in area and power consumption [144].

To reduce the critical path, the Sweeney [145], Robertson [146] and Tocher [147] (SRT) algorithm speculates a quotient bit based on a few MSBs of the divisor and the partial remainder.

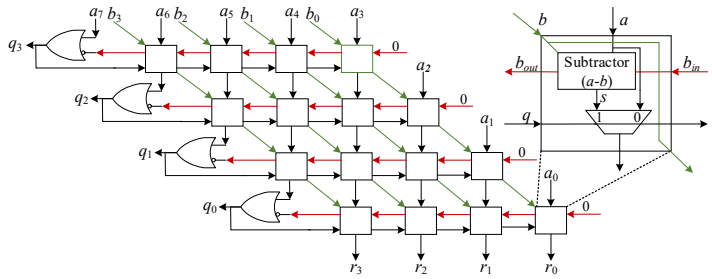


Fig. 18. An 8/4 unsigned restoring array divider. $A=\sum_{i=0}^{7} a_i 2^i$ and $B=\sum_{i=0}^{b} b_i 2^i$ are the input dividend and divisor, respectively. $Q=\sum_{i=0}^{3} q_i 2^i$ and $R=\sum_{i=0}^{3} r_i 2^i$ are the output quotient and remainder, respectively. The OR gate is a simplified subtractor with 0 as the subtrahend for dealing with overflow.

In an SRT divider, therefore, the bit width of the subtractors is smaller than that in an array divider, thus resulting in a faster operation. The performance can be further improved by using a high-radix divider that generates several quotient bits rather than one at each iteration [148]. The quotient in an SRT division is usually in a redundant form, so an on-the-fly conversion algorithm [149] is required to convert the quotient into a non-redundant representation. Notably, the performance of these dividers is improved by trading off circuit area and power consumption.

Several iterative algorithms, including the Newton-Raphson [10] and Goldschmidt [9] algorithms, have been developed for large division using multiplication and addition. The performance of this type of dividers are significantly affected by the presumed initial parameter values.

Several approximate subtractor/adder cells have recently been proposed for an array divider [31], [150]–[152]. Another type of approximate dividers uses a reduced-width exact divider for large division [39], [153], while several others are based on functional approximation (e.g., using a logarithmic algorithm) [30], [33], [154]–[156] and curve fitting [157].

B. Review

1) Approximation in subtractor/adder cells: In [31], three approximate subtractors obtained by simplifying the circuit of an exact cell are used for processing some LSBs in a vertical, horizontal, square or triangle region in an array divider. This design is referred to as an approximate unsigned non-restoring divider (AXDnr). Compared to the AXDnrs, similarly-designed approximate restoring dividers (AXDrs) show better tradeoffs with slightly higher accuracy and lower power dissipation [150].

In [151], an approximate signed-digit adder is proposed for use in high-radix dividers, together with replacement, truncation and error compensation in an array structure. For an array divider, a high-radix design can be faster but consumes more power compared to a radix-2 design [152].

2) Using a reduced-width exact divider: A dynamic approximate divider (DAXD) selects the inputs and uses a reduced-width restoring array divider [153], in a similar way to the design of DRUM [120]. This selection scheme could lead to overflows that cause a low accuracy in division. To implement a DAXD, two leading-one detectors, two multiplexers, a reduced-width array divider, a subtractor and a barrel shifter are needed.

Depending on the positions of the leading "1"s, an adaptively approximate divider (AAXD) employs two pruning schemes to determine the inputs for a reduced-width exact divider [39]. Different from the DAXD, zeros are appended to the LSBs for selecting k input bits when the leading "1" is within the k LSBs. In addition, an error correction unit is used to ensure a high accuracy with a very low maximum error distance.

3) Approximate dividers based on functional approximation: In the high-speed and energy-efficient approximate divider (SEERAD) [154], the division is implemented by a simple multiplication by rounding the divisor B to a form of $2^{K+L}/D$, where K indicates the leading "1" position of B, and L and D are constant integers estimated via an exhaustive simulation for achieving the lowest mean relative error. For a division of A/B, it is then sufficient to use a multiplier for computing AD, a barrel shifter, and some lookup tables for storing L and D. Different accuracy levels are obtained by varying D and L.

A binary logarithm-based functional approximation performs division by computing the antilogarithm of the difference between the logarithmic values of the dividend A given by (5), and divisor B given by (6). It leads to

$$\log_2 Q = \log_2 (A/B) \approx k_1 - k_2 + x_1 - x_2, \tag{11}$$

where k_1 and k_2 specify the leading one positions of A and B respectively, $x_1 = A/2^{k_1} - 1$, and $x_2 = B/2^{k_2} - 1$.

Using Mitchell's algorithm, an approximate division can be implemented by performing the antilogarithm of (11), i.e.,

$$Q \approx \begin{cases} 2^{k_1-k_2}(x_1 - x_2 + 1) & \text{if } x_1 - x_2 \geq 0 \\ 2^{k_1-k_2-1}(x_1 - x_2 + 2) & \text{if } x_1 - x_2 < 0 \end{cases}. \tag{12}$$

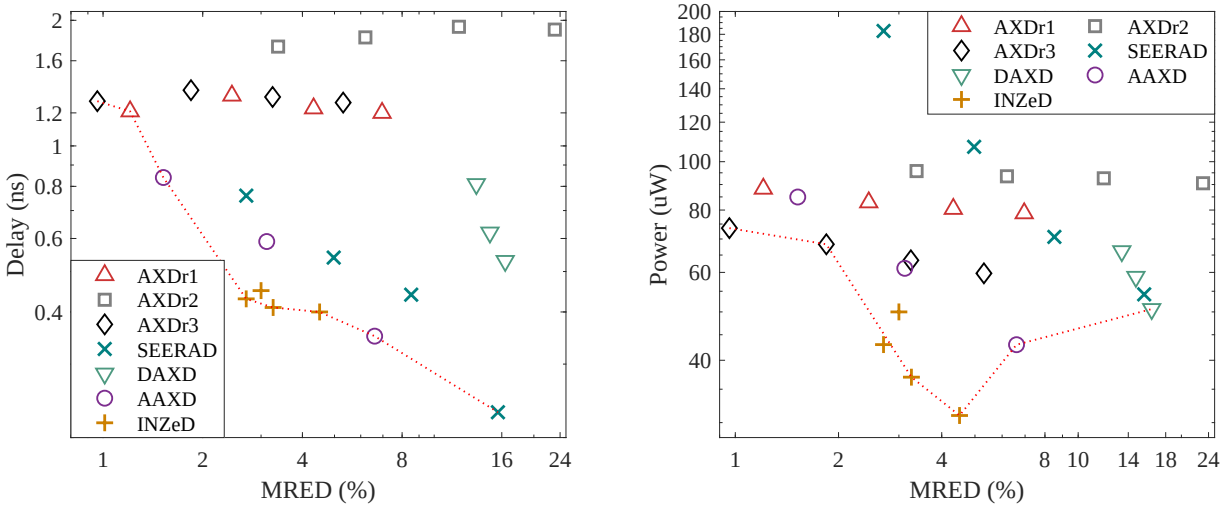
Error correction is considered in [33] to compensate an offset to the computed quotient. Additionally, a number of LSBs are truncated in the subtractors for implementing (12). These techniques enable the design of approximate integer and floating-point dividers with near-zero error bias, denoted as INZeD and FaNZeD, respectively.

In a high-speed divider (HSD) [30], a piecewise linear approximation is utilized to implement the antilogarithm directly on the two input operands, thus only lookup tables and multiplications are required. Compared to a divider implemented using Mitchell's algorithm, the HSD is more accurate and faster with a larger area.

An approximate hybrid divider (AXHD) is based on a restoring array structure and logarithmic approximation [155]. In this design, the p MSBs in a $2n/n$ divider is accurately implemented as a restoring array divider, while the $(2n-p)$ LSBs are approximately processed using Mitchell's algorithm as per (12).

In [156], the mantissa in floating-point division is approximately computed by a subtractor; the approximation is then tuned by using an error compensation lookup table (storing precomputed values) and a subtractor. This design is denoted as the configurable approximate divider for energy efficiency (CADE) as its accuracy varies with the size of the lookup table.

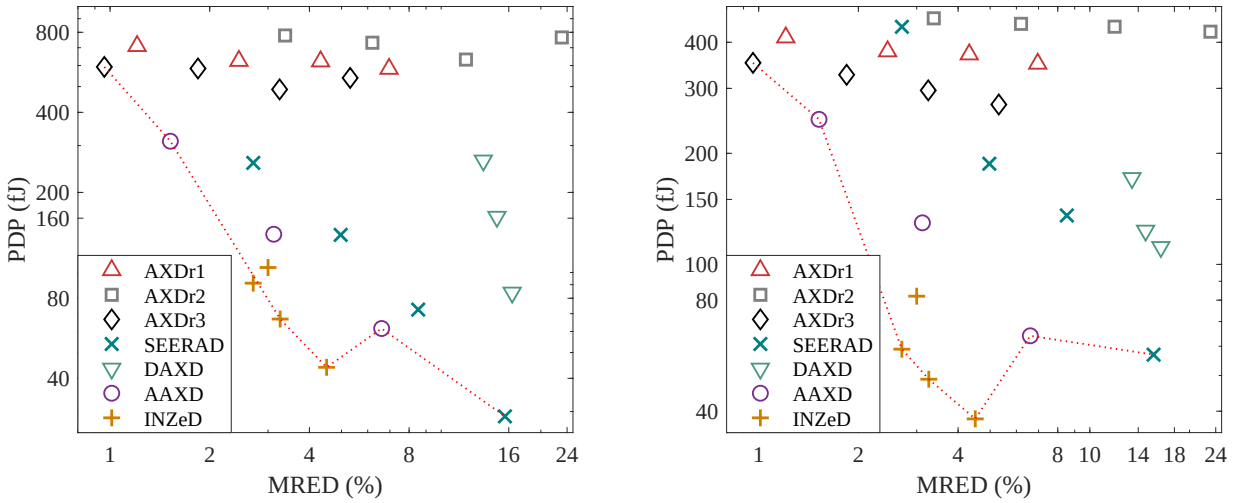
4) Curve fitting based approximate dividers: In the design of a floating-point divider, the curved surfaces of the quotient are partitioned into several square or triangular regions that are linearly approximated by curve fitting [157]. The mantissa division is then implemented by a comparison module, a lookup table, shifters and adders. With a similar circuit structure to the HSD, this approximate divider achieves a higher accuracy.



(a) Delay (delay-optimized) vs. MRED

(b) Power (area-optimized) vs. MRED

Fig. 19. Optimized delay and power consumption vs. MRED for the approximate 16/8 unsigned integer dividers. *Note*: The replacement depths of AXDr1, AXDr2 and AXDr3 are from 8 to 11 from left to right. The accuracy levels of SEERAD are from 4 down to 1 from left to right. The dividend width after pruning is from 12 down to 8 with a decrement of 2 for DAXD and AAXD from left to right. In INZeD, the number of LSBs truncated in the subtractor is 0, 2, 3 and 4, from left to right.



(a) PDP (delay-optimized) vs. MRED

(b) PDP (area-optimized) vs. MRED

Fig. 20. A comparison of the power-delay product and MRED for the approximate 16/8 unsigned integer dividers.

C. Evaluation

In this evaluation, we consider approximate 16/8 unsigned integer dividers, including AXDr, DAXD, AAXD, SEERAD and INZeD. The high-radix and floating-point dividers are not considered. Three designs of AXDr with the triangle replacement (that shows the best tradeoff [150]) are selected for evaluation, i.e., AXDr1, AXDr2 and AXDr3 using three different approximate subtractors. For DAXD and AAXD, the reduced-width exact dividers are implemented using an array structure. An exhaustive simulation is performed for the error evaluation, i.e., all valid combinations in the range of $[0, 65535]$ and $(0, 255]$ that do not cause overflow in an accurate 16/8 divider, are used as the input dividends and divisors. The same tools, technologies and configurations as for the evaluation of adders and multipliers are applied here. The clock period for the power estimation is set to 5 ns. As MRED and NMED show a similar pattern in the simulation results, only MRED is plotted in Fig. 19(a) and (b), respectively, against which the optimized delay and power consumption for area-optimized synthesis are shown. Fig. 20 presents the comparison in PDP versus MRED for both delay- and area-optimized syntheses.

Hardware vs. accuracy: As can be seen in Fig. 19, AXDr1 and AXDr3 can be very accurate with a moderate power consumption, but quite slow, whereas DAXD is the least accurate in general. For a medium-low MRED, AAXD and INZeD show the highest performance and the lowest power consumption; thus, they achieve the best accuracy-hardware tradeoff. This is also evident in the PDP and MRED figure in Fig. 20. SEERAD is the fastest at a low accuracy (Fig. 19) and with the lowest PDP for delay-optimized synthesis (Fig. 20).

TABLE IV. Summary of approximate 16/8 unsigned integer divider designs.

Divider	Error characteristics	Circuit measurements		
	MRED	Performance (delay-optimized)	Power (area-optimized)	PDP
AXDr1	low	low		high
AXDr3	low	low		high
SEERAD-1	high	high		low
AAXD	low	high		low
INZeD		high	low	low

In summary, AAXD is an efficient design for applications that require a high accuracy and high performance. Although some configurations of AXDr1 and AXDr3 are very accurate, they are generally slow with high energy consumption. INZeD is the most efficient design for a moderate accuracy. For an application that can tolerate a high level of inaccuracies, SEERAD-1 is suitable with a low hardware cost. A qualitative summary of these features is shown in Table IV.

VI. Applications

A. Image Processing

To assess the capabilities of the approximate designs, we consider three image processing applications: image sharpening using unsigned multipliers and adders, image compression using signed multipliers and adders, and change detection using unsigned dividers.

1) Image sharpening: This technique enhances the edges in an image to obtain a clearer view. An image with a pixel matrix $\mathbf{I}$ is sharpened by using $\mathbf{R}(x,y) = 2\mathbf{I}(x,y) - \mathbf{S}(x,y)$ [158], where $\mathbf{R}(x,y)$ is a resultant image pixel, and $\mathbf{S}(x,y)$ is obtained by a convolution,

$$\mathbf{S}(x,y) = \frac{1}{273}\sum_{m=-2}^{2}\sum_{n=-2}^{2}\mathbf{G}(m+3,n+3)\mathbf{I}(x-m,y-n), \quad (13)$$

where $\mathbf{G}$ is a 5×5 convolution matrix given by

$$\mathbf{G} = \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}. \quad (14)$$

Equation (13) shows that 25 multiplications, 24 additions and a division are required for computing $\mathbf{S}(x,y)$. In this simulation, the inputs are normalized by the maximum numbers and scaled to a number in 16-bit unsigned integer representation, so 16×16 approximate unsigned multipliers and 16-bit approximate adders are used to implement the sum of products in (13). Note that the 32-bit products are rounded to 16 bits as inputs to the adders. The division by 273 is implemented by a multiplication of a constant input 1/273.

Among the approximate adders and multipliers in each category, one or two designs with the best accuracy and energy tradeoffs are selected for this application. Hence, ACA, GeAr, ETAII, CCBA, LOA and TruA are considered for addition; UDM, BAM, DRUM, HOCM, TAM1, ICM, ALM-SOA, CGPM3 and TruMW are selected for multiplication. The configurations that lead to similar PDPs compared to other designs are considered for designs with variable parameters.

Fig. 21 shows the PSNRs of the sharpened images, which are infinite for the combinations of the accurate adder (AccuA) and accurate multiplier (AccuM), and AccuA and ICM. The input image is a blurred *"Lena"* with 512×512 pixels. Because ICM has a very low ER (5.45%), and the error does not or seldom occur in this application, the ICM is as effective as AccuM for image sharpening. For the same reason, the images sharpened by using the UDM have very close PSNRs to the ones processed by an accurate multiplier. These results illustrate the advantages of designs with low ERs in certain applications.

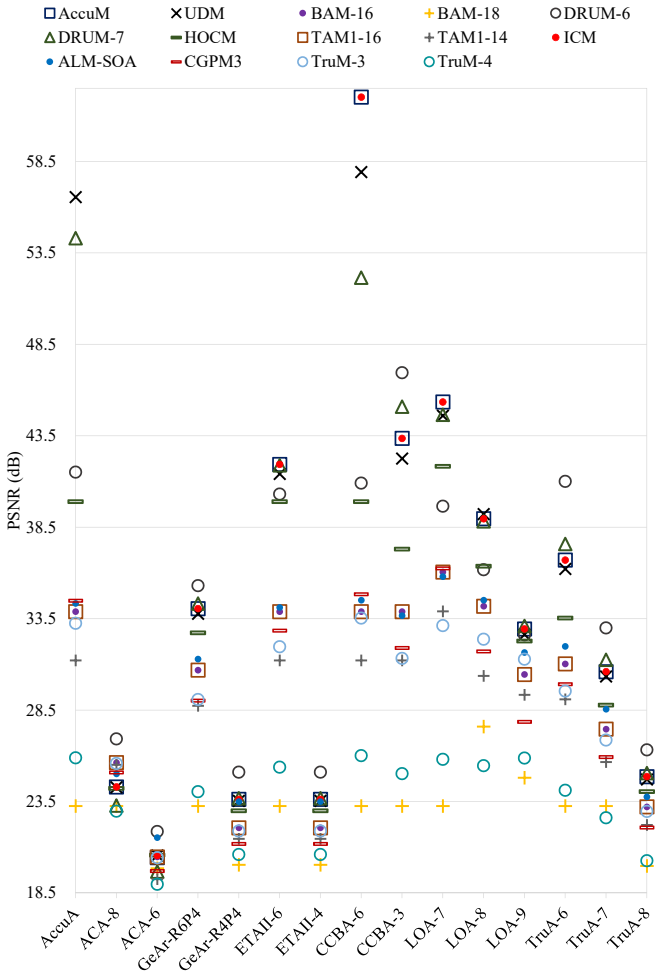


Fig. 21. A PSNR comparison of image sharpening results. *Note*: AccuA and AccuM denote the accurate adder and multiplier, respectively. The numeric value in the name of each design indicates the parameter value. The design 1StepTrunc is considered for HOCM; the configuration of 280 is selected for CGPM3.

Although DRUM-7 and DRUM-6 show larger values of MRED and NMED than the other designs, they lead to higher PSNRs due to their unbiased errors. Also, HOCM results in images with a higher quality than many other multipliers, when used with a same adder. With a larger MRED and NMED, ALM-SOA performs similarly to BAM-16 and TAM1-16, because BAM and TAM1 generate single-sided errors that can be accumulated in the sum of products.

When an approximate adder (or multiplier) has a very low accuracy (e.g., BAM-18, TruM-4, ACA-6, GeAr-R4P4 and ETAII-4), increasing the accuracy of the collaborating multiplier (or adder) does not improve the quality of the processed image. It is worth noting that using some combinations of approximate adders and multipliers can lead to a higher image quality than

using solely an approximate adder or an approximate multiplier. For example, the use of LOA-7 and BAM-16 results in an image with a higher PSNR than BAM-16 and AccuA, and the combination of CCBA-3 and DRUM-6 outperforms the duo of CCBA-3 and AccuM.

The circuit designs for image sharpening are synthesized for optimized area (the same for the other applications). The clock period for the power estimation is 10 ns. In the simulation, the CLA and Wallace multiplier are utilized for the AccuA and AccuM, respectively. Fig. 22(a) shows that the implementations using TAM1-16 are the fastest, followed by HOCM, whereas the ones using DRUM-6 are the slowest, followed by ICM. The delay values are not as consistent as the area results for different adder and multiplier combinations because the syntheses are optimized for area.

As shown in Fig. 22(b) and (c), using different adder designs does not significantly affect the area or power dissipation when a specific multiplier is used. Thus, the multiplier dominates the area and power dissipation for this application. On the contrary, the adder plays a more significant role on the critical path delay, as shown in Fig. 22(a), because the 25 multipliers work in parallel, whereas the 24 adders work in a tree structure, resulting in a critical path of one multiplier and five adders. Among the multipliers, ALM-SOA and TAM1-16 are very energy-efficient, as shown in the PDP values in Fig. 22(d).

Fig. 23 shows the comparison of PDP reductions of different implementations compared with the accurate design, in the descending order of PSNRs larger than 30 dB. For an implementation that produces sharpened images with a PSNR higher than 35 dB, ETAII-6 and HOCM lead to the most significant saving in PDP (by about 50%). For a PSNR between 30 dB and 35 dB, LOA-9 and ALM-SOA are the most efficient with 69% reduction in PDP. DRUM-6 achieves the largest reductions in PDP for a high image quality (with a PSNR larger than 40 dB), whereas ALM-SOA is the most efficient for a relatively low image quality (with a PSNR lower than 35 dB).

2) JPEG compression: Based on the discrete cosine transform (DCT), JPEG is a widely used lossy compression algorithm for digital images [159]. The image pixels in the spatial domain are first converted into the frequency domain via a DCT. In the DCT, the pixel matrix of an image in 16-bit 2's complement is divided into 8×8 blocks. Each block $\mathbf{B}$ is converted to the frequency domain by

$$\mathbf{D} = \mathbf{T}\mathbf{B}\mathbf{T}', \tag{15}$$

where $\mathbf{T}$ is an 8×8 DCT coefficient matrix given by

$$\mathbf{T}(x,y) = \begin{cases} \frac{1}{\sqrt{8}} & \text{if } x = 0 \\ \frac{1}{2}\cos\left[\frac{(2y+1)x\pi}{16}\right] & \text{if } x > 0 \end{cases}, \tag{16}$$

where $x = 0, 1, \cdots, 7$ and $y = 0, 1, \cdots, 7$.

The high-frequency information is then discarded by the quantization

$$\mathbf{C}(x,y) = \text{round}\left(\frac{\mathbf{D}(x,y)}{\mathbf{Q}(x,y)}\right), \tag{17}$$

where $\mathbf{Q}$ is a quantization matrix of unsigned integers determined by the required quality level. The quality level can be from 1 to 100, where 1 corresponds to the highest compression ratio and thus the poorest image quality.

To reconstruct the image, the above operations are inverted by de-quantization and inverse DCT (IDCT):

$$\mathbf{R}(x,y) = \mathbf{C}(x,y) \times \mathbf{Q}(x,y), \tag{18}$$

and

$$\mathbf{I} = \mathbf{T}'\mathbf{R}\mathbf{T}. \tag{19}$$

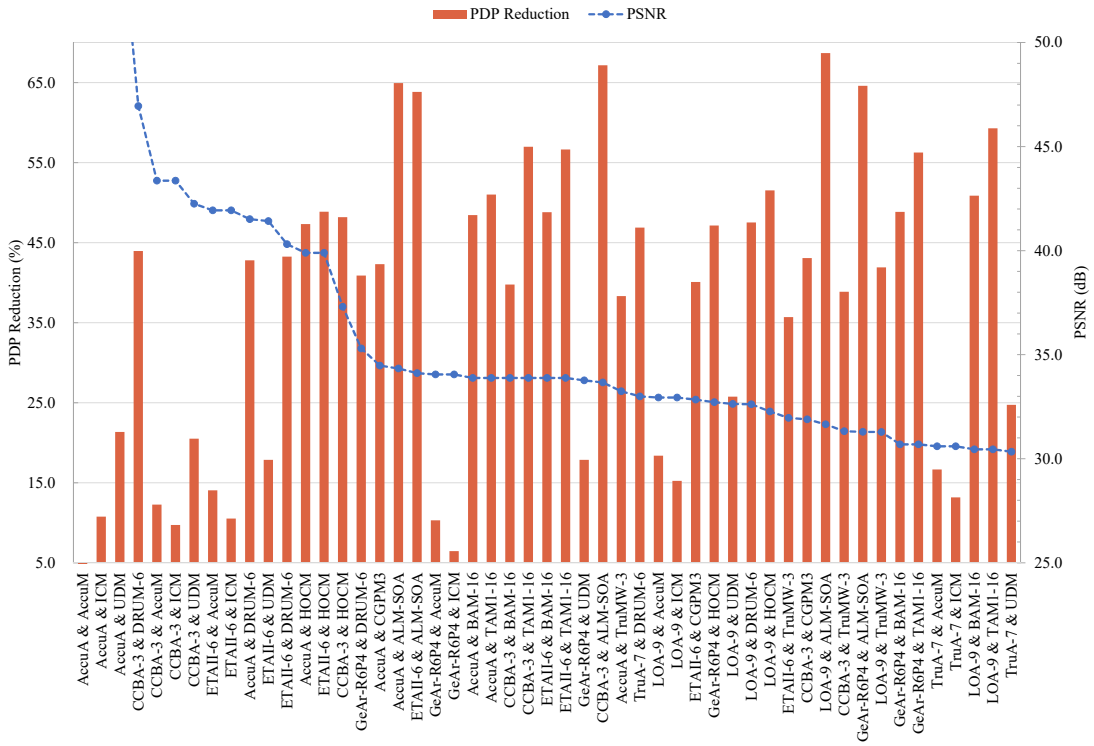


Fig. 23. A comparison of PDPs in the descending order of PSNRs for various adder-multiplier implementations of image sharpening.

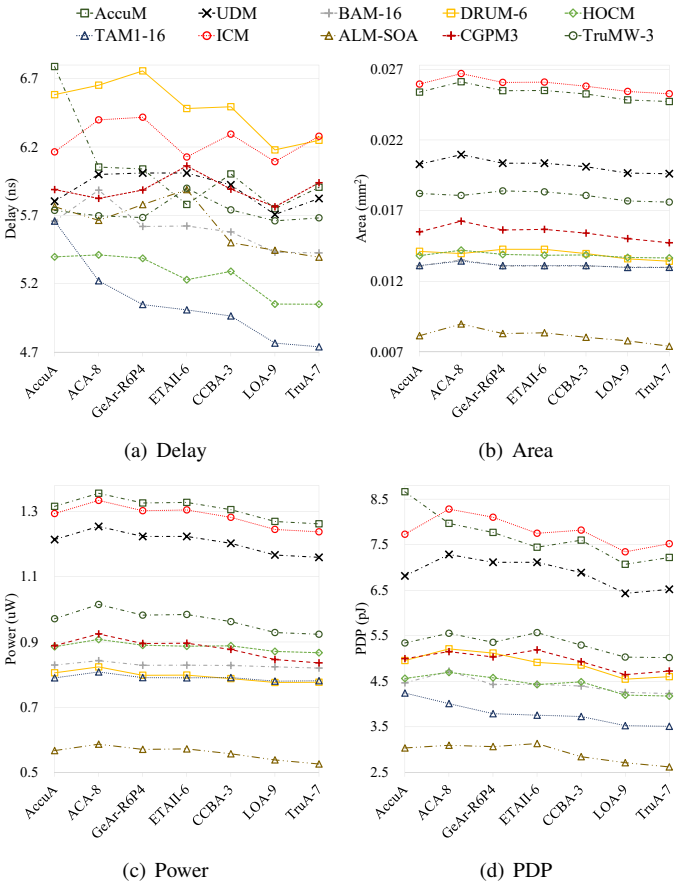


Fig. 22. Circuit measurements of image sharpening.

In this simulation, the signed coefficients in matrix **T** are scaled to 16-bit 2's complement format, and the image pixels are normalized by the maximum number followed by a subtraction of 0.5 and scaled to 16-bit 2's complement format. The signed multiplications in the DCT and IDCT are implemented by approximate Booth multipliers, including the 16×16 designs with good tradeoffs in accuracy and hardware, BM07, PEBM, ABM2, BBM and TBM. The same 16-bit adder designs as in the image sharpening are used. The quality level for the compression is 50.

The qualities of the decompressed images using different adder and multiplier combinations are shown in Fig. 24. As can be seen, ACA, GeAr and ETAII are not suitable for this application although they have very low ERs. Note that the errors of these approximate adders are single-sided (or negative) due to the dropping of some carry bits; thus, the error biases for these types of adders are very large [51]. As a result, errors are accumulated in the multiple matrix multiplications and cannot be tolerated in DCT or IDCT. Similarly, BBM has a larger error bias than the other approximate Booth multipliers due to the truncation of the partial products. Thus, the use of BBM also produces images with a low quality in most cases. Among the approximate adders, LOA-3 performs the best, followed by TruA-1, while for the approximate multipliers, BM07, PEBM, and TBM-2 outperform the other designs when a same adder is used.

It is worth to note that using some approximate Booth multipliers along with some approximate adders, e.g., PEBM and TruA-1, PEBM and TruA-2, generates a significantly higher

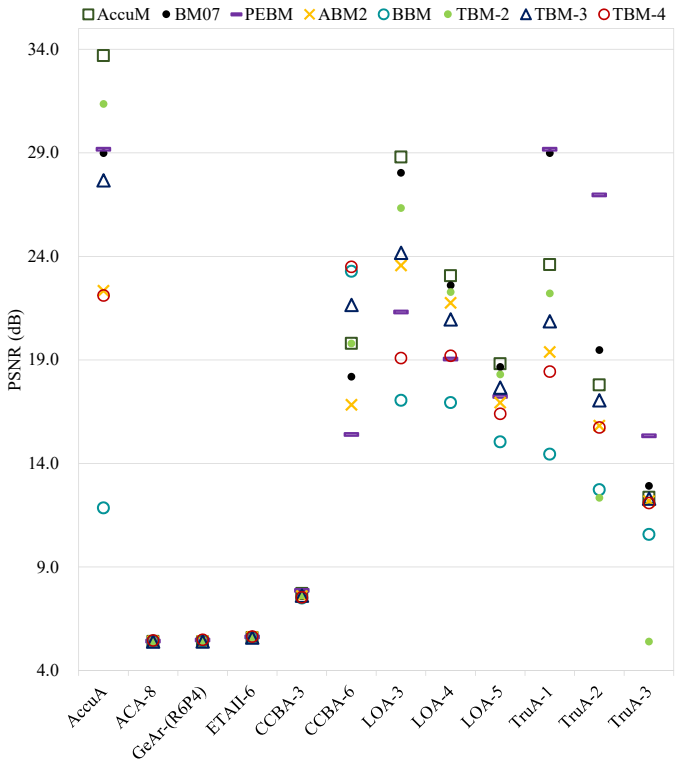


Fig. 24. A comparison of JPEG compression and decompression quality using different adder and multiplier designs.

quality than the other designs (even when an accurate multiplier is used). Except for these special cases, the image quality in PSNRs increases with the decrease in the MREDs of the utilized approximate Booth multipliers.

Additionally, the results in Fig. 24 shows that a more complex computation involving multiple matrix multiplications is more sensitive to errors in addition than those in multiplication with the same bit width. Thus, a larger approximation can be tolerated in multiplication than in addition. The tolerable approximation in addition is to a lesser extent in JPEG compression than in image sharpening for an acceptable accuracy, e.g., LOA-3 is required for JPEG compression while LOA-8 is sufficient for image sharpening.

Fig. 25 shows the resulting PDP reductions of the DCT implementations using different multiplier and adder combinations compared with the accurate design. The clock period used for the power estimation is 10 ns. The accurate DCT design utilizes a 16-bit post-truncated fixed-width Booth multiplier and a 16-bit CLA. As shown in Fig. 25, the combination of AccuA and PEBM shows the best tradeoff in this implementation, achieving the highest PDP reduction (about 20%) with a relatively high PSNR (nearly 30 dB). Using an approximate adder with PEBM rather significantly degrades the image quality. Among the approximate Booth multipliers, PEBM, TBM-3 and ABM2 lead to higher energy efficiency than the other designs for this application.

3) Change detection: The changes in two images can be detected by finding the ratios between the corresponding pixels. Thus, change detection can be used to assess the approximate dividers. In each design, one configuration is selected to ensure that a similar PDP (e.g., AXDr1-11, AXDr2-11, AXDr3-11, DAXD-8, AAXD-8, SEERAD-1 and INZeD-2), or MRED (e.g., AXDr1-9, AXDr2-8, AXDr3-10, DAXD-12, AAXD-10,

TABLE V. Change detection results using different dividers.

Accurate divider	AXDr1-11 (28.82 dB)	AXDr2-11 (18.09 dB)	AXDr3-11 (33.72 dB)	DAXD-8 (25.22 dB)	AAXD-8 (34.95 dB)	SEERAD-1 (22.08 dB)	INZeD-2 (37.34)
	AXDr1-9 (35.67 dB)	AXDr2-8 (22.91 dB)	AXDr3-10 (35.94 dB)	DAXD-12 (23.56 dB)	AAXD-10 (40.16 dB)	SEERAD-4 (36.61 dB)	INZeD-0 (33.79 dB)

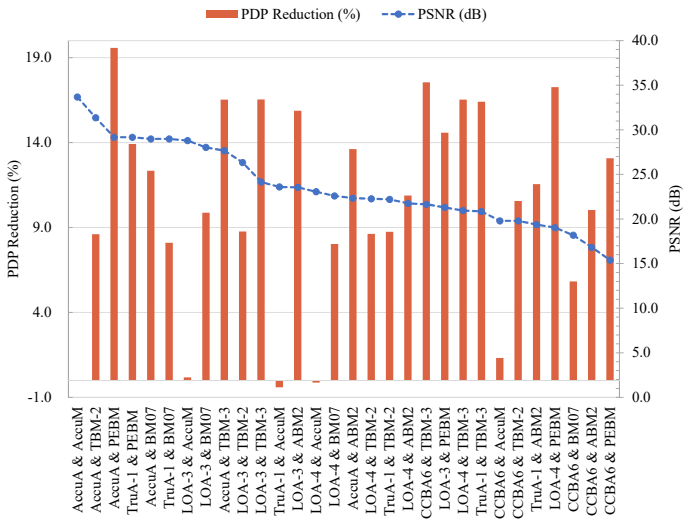


Fig. 25. A comparison of PDPs in the descending order of PSNRs for DCT implementations using different adder and multiplier designs.

SEERAD-4 and INZeD-0) occurs for the considered designs. The 16/8 unsigned integer dividers are utilized to obtain the pixel ratios. As shown in Table V, AXDr1-9, AXDr3-10, AXDr3-11, AAXD-10, INZeD-0, INZeD-2 and SEERAD-4 perform similarly well as an accurate divider, whereas AXDr2-11, DAXD-12, DAXD-8 and SEERAD-1 produce results with a lower quality. AXDr2-8 and AAXD-8 produce images that are acceptable for a visual inspection. As the implementation of change detection mainly consists of dividers, the circuit measurements are similar to those for approximate dividers; thus, they are not shown here. Fig. 20 shows that to achieve a result with a PSNR higher than 33 dB, INZeD-2 consumes the smallest energy followed by INZeD-0 and AAXD-10.

B. Deep Neural Networks

Face detection and alignment are two common tasks in machine learning. Using DNNs, the accuracy of face detection and alignment have been significantly improved since the early 2010s. Due to the correlation of these two tasks, a multi-task CNN (MTCNN) has been proposed for joint face detection and alignment [160]. An accelerator specifically designed for this MTCNN achieves a high energy efficiency and throughput [161]. In this MTCNN, three CNNs cascade as the proposal network (P-Net), the refine network (R-Net) and the output network (O-Net). The basic operation in a CNN is the convolution based on multiplications and additions.

To assess the viability of approximate circuits in DNNs, the 16×16 approximate Booth multipliers and 16-bit adders are integrated into the architecture of an MTCNN for face detection and alignment, as shown in Fig. 26. Here, the convolutional (CONV) layers account for the most computations. The max pooling is used for all pooling layers. Two fully connected (FC) layers to the end of the R-Net and P-Net are implemented by vector multiplications. The approximate Booth multipliers with good tradeoffs in accuracy and hardware, as those used in the JPEG compression, are considered in this application. One approximate adder with single-sided errors (i.e., ETAII), one with a small error bias (i.e., LOA), and the truncated adder (TruA) are selected for additions.

Fig. 27 shows some face detection (in the bottom row) and face alignment (in the top row) results using different adders and multipliers. For the face detection, a square is drawn to show the detected area. To align a face, five landmarks are used to mark the eyes, nose and mouth. Compared to the accurate implementation using AccuM and AccuA, BM07 and ETAII-7 perform poorly in face detection and alignment, as indicated by the squares and landmarks far away from the target positions, whereas BM07 and LOA-4 result in a better quality.

To quantitatively assess the accuracy of the face detection, the true positive rate (TPR) is measured for each implementation on the FDDB dataset [162], as shown in Table VI. The TPRs in Table VI show that the approximate Booth multipliers (except for BBM), when working with LOA-3, LOA-4 and TruA-1, perform well in face detection, resulting in close TPRs to the accurate implementation. ETAII-7 leads to very small TPRs due to its large error bias. Similarly, TruA-2 (except for the combination with PEBM) and BBM (except for the combinations with LOA-3 and LOA-4) result in relatively low TPRs. Interestingly, ABM2 and TBM-4 working with the accurate adder result in higher TPRs than the accurate design. Similar results have been observed in [47].

In addition, the number of multiply-and-accumulates (MACs) required to detect the faces in one image averaged over the FDDB dataset is reported in Table VII as an indicator for energy efficiency. Overall, ABM2 and LOA-3, AccuM and TruA-1, BM07 and TruA-1, PEBM and TruA-1, and PEBM and TruA-2, are effective combinations for face detection, which result in high TPRs and require a smaller number of MACs (thus, a higher energy efficiency) than the accurate implementation. Hence, the energy efficiency of a DNN can be improved by using approximate arithmetic circuits while achieving a similar or even higher detection accuracy.

Finally, the normalized mean errors (NMEs) for the face align-

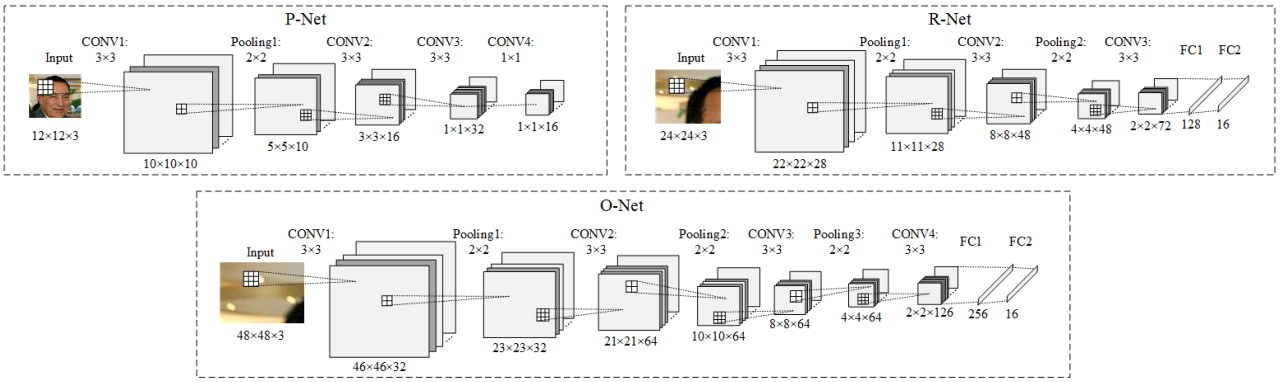
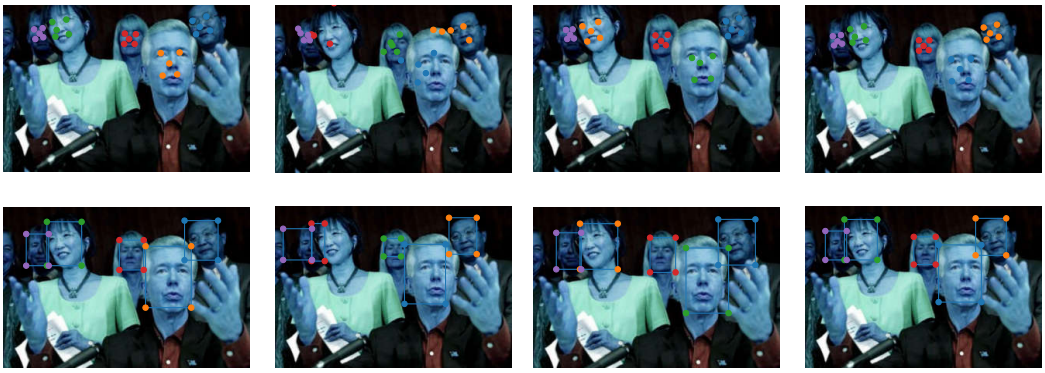


Fig. 26. The architecture of the implemented MTCNN.



(a) AccuM & AccuA (b) BM07 & ETAII-7 (c) BM07 & LOA-4 (d) TBM-4 & TruA-1

Fig. 27. Some face detection (at bottom) and alignment (on top) results using different adders and multipliers.

TABLE VI. True positive rates of the face detection on FDDB dataset (%). The values higher than 90% are highlighted in bold, and the ones larger than that of the accurate design are in red.

Design	AccuA	ETAII-7	LOA-3	LOA-4	TruA-1	TruA-2
AccuM	**91.26**	54.07	**90.21**	89.96	**90.27**	72.13
BM07	**90.79**	66.58	89.77	89.75	**90.99**	84.70
PEBM	**90.47**	77.64	81.98	80.56	**90.56**	**90.18**
ABM2	**91.34**	46.47	**90.10**	**90.25**	88.40	57.57
BBM	83.77	23.03	89.07	89.17	61.52	10.25
TBM-3	**91.04**	53.97	**90.68**	**90.56**	**90.31**	71.55
TBM-4	**91.34**	52.33	**90.79**	**90.91**	**90.01**	71.49

TABLE VII. The number of MACs required to detect the faces in one image averaged over the FDDB dataset (in billions). The numbers smaller than that of the accurate design are in bold.

Design	AccuA	ETAII-7	LOA-3	LOA-4	TruA-1	TruA-2
AccuM	0.5817	0.6583	0.5893	0.5962	**0.5802**	0.5826
BM07	0.5882	0.6365	0.5822	0.5890	**0.5740**	**0.5742**
PEBM	0.5831	0.6336	**0.5716**	**0.5780**	**0.5655**	**0.5675**
ABM2	0.5907	0.7647	**0.5794**	0.5834	**0.5629**	**0.5758**
BBM	0.6046	0.6786	0.6048	0.6167	0.6086	0.6179
TBM-3	0.5867	0.6647	0.5929	0.6019	0.5861	0.5896
TBM-4	0.5915	0.6709	0.5969	0.6049	0.5906	0.5957

ment are obtained by comparing the coordinate values of the five landmarks with their standard values for the AFLW dataset [163], as shown in Table VIII. The combinations of designs that result in small TPRs are omitted. It is interesting that the MTCNNs using LOA-3 and LOA-4 consistently achieve smaller NMEs than those using accurate adders. Although TruA-1 performs well in face detection, it results in large NMEs in face alignment, LOA-3 performs the best among the approximate adders. Among the approximate multipliers, BM07 and PEBM are effective

TABLE VIII. Normalized mean errors for the face alignment on AFLW dataset (%). The errors smaller than that of the accurate design are highlighted in bold.

Design	AccuA	LOA-3	LOA-4	TruA-1
AccuM	5.379	**3.145**	**3.520**	9.920
BM07	**4.393**	**3.063**	**3.624**	8.699
PEBM	**3.050**	**2.988**	**3.551**	9.648
ABM2	5.743	**3.204**	**3.570**	10.49
TBM-3	5.718	**3.171**	**3.667**	10.25
TBM-4	5.761	**3.241**	**3.667**	10.41

designs producing smaller NMEs than the accurate design. Note that the approximate adders and multipliers that result in low accuracy in face detection and alignment (BBM, ETAII-7, TruA-1 and TruA-2) share a same feature, single-sided errors. Similar to the JPEG compression, the face detection and alignment are more sensitive to errors in additions than in multiplications (with the same bit width), so a deeper approximation can be tolerated in approximate multipliers.

VII. Conclusions, Challenges and Prospects

In this article, approximate arithmetic circuits are reviewed, characterized and comparatively evaluated, using functional simulation, circuit synthesis optimized for performance and area, and image processing and machine learning applications.

A. Characterization

Approximate Adders: Most of the approximate adders have been designed for high performance and low error (in ER) by reducing the critical path delay. Most speculative adders,

segmented adders and carry-select adders show low ERs. Due to the reduction of some carries, single-sided errors are prevalent in these designs that result in large error biases, especially in applications that require iterative or repetitive additions. However, the designs using approximate full adders in the LSBs often have high ERs, low MREDs and low power dissipation. With a reduced precision, a truncated adder produces a biased error with a high ER close to 100%, but it consumes a very low power. Considering practically-effective error and circuit metrics such as MRED and PDP, LOA, CCBA and the truncated adders achieve the best accuracy-energy tradeoffs.

Approximate Multipliers: For unsigned designs, truncating part of the partial products or some LSBs of the input operands is an effective scheme to reduce circuit area, while preserving a moderate and variable accuracy in terms of NMED and MRED, depending on the number of bits truncated; examples include the BAM, TAM1 and even, the truncated multipliers. Logarithmic multipliers are relatively inaccurate, but they can be very efficient in performance and power consumption. The truncated Wallace multiplier, compressor-based HOCM and TAM1 are among the designs with a high performance, while the truncated array multiplier is more power efficient at a moderate accuracy. The CGPMs can be very accurate with a moderate performance. TAM1, HOCM and the logarithmic design ALM-SOA show the best tradeoffs between energy and accuracy. For signed multipliers, most fixed-width Booth multipliers provide a better design tradeoff than the truncated Booth multipliers due to the efficient error compensation.

Approximate Dividers: For the fewer divider designs, those approximated in the subtractor/adder cells are slow, and their accuracy varies with the approximate subtractor/adder design. The dividers based on functional approximation are relatively fast. Among the considered designs, the logarithmic INZeD and input-adaptive AAXD provide balanced tradeoffs with both low PDPs and MREDs.

The above observations are based on the investigation of 16-bit designs, so the circuit and error characteristics may vary for adders of different sizes, although some adders are based on regular structures. Multipliers and dividers of different sizes may exhibit more significantly different characteristics as some designs are tailored and optimized for a specific bit-width; thus, the performance may degrade even though the approximation scheme is scalable.

In general, rather limited improvements in circuit measurements are observed for the approximate arithmetic circuits simplified from an accurate design. Many ad hoc designs underperform simply truncated circuits. A functional approximation algorithm such as the binary logarithm can lead to designs with significant savings in circuit area and power consumption, albeit at the cost of a low accuracy. With the potential of breaking away from the original (limiting) architecture, nevertheless, functional approximation might be promising for hardware-efficient approximate arithmetic circuits, though leaving the challenge of enhancing its accuracy with low hardware overhead.

B. Applications

For image processing, the approximate adders, multipliers and dividers with smaller error magnitudes (in MREDs) generally produce results with a higher quality. For simple operations such as the sum of products, the approximate multipliers with lower ERs outperform those with higher ERs. Although with very low ERs, the approximate adders with large error biases (e.g., ACA, ETAII, TruA) do not work well for more complex computations such as cascaded matrix multiplications. In an accumulative operation, the approximate designs with low error biases or double-sided errors consistently perform better than those with single-sided errors.

The more complex computation that involves multiple matrix multiplications is more vulnerable to errors in addition than those in multiplication. In other words, a larger approximation can be tolerated in multipliers than in adders to achieve a reasonably accurate result. In such applications, the multiplier dominates the area and power dissipation of the circuit, whereas more adders are in the critical path, so the adder plays a more important role on the delay.

By using approximate adders and multipliers, an MTCNN can achieve a comparable face detection quality to the accurate design. The accuracy of the face detection generally decreases with the increase of the MREDs for the approximate multipliers, so the MRED is an indicator of the quality of an approximate multiplier. For this more complex application, the approximate adders and multipliers with large error biases result in significantly poor accuracy in face detection and alignment. Compared to the accurate design, a smaller number of MACs is required for face detection when some approximate designs are used in an MTCNN. Hence, the use of approximate arithmetic circuits can reduce the power consumption and improve the energy efficiency of an MTCNN.

Interestingly, some combinations of approximate multipliers and adders lead to higher accuracy than the accurate implementation, even though these circuits, by themselves, do not show advantages over the others. Hence, it might be more effective to design approximate arithmetic circuits from a system's or application's perspective. A rigorous evaluation framework with trustworthy error metrics would be imperative to ensure the reliability and robustness of the system with respect to the effect due to the propagation and statistical distributions of errors. Approximate arithmetic circuits could also be integrated with other approximate components in a system hierarchy, such as memory and interconnects, for a more significant improvement in hardware efficiency as well as processing quality.

ACKNOWLEDGMENT

This work was supported by the China Postdoctoral Science Foundation under Grant 2019M650679, the National Natural Science Foundation of China (Grant No. 61834002), the National Key R&D Program of China (Grant No. 2018YFB2202101), the National Science and Technology Major Project of the Ministry of Science and Technology of China (Grant No. 2018ZX01027101-002), the National Council of Science and Technology (CONACYT) and Mexican Foundation for Education, Technology and Science (FUNED) and the Natural Sciences and Engineering Research Council (NSERC) of Canada under Project RES0025211.

REFERENCES

[1] J. Han and M. Orshansky. Approximate computing: an emerging paradigm for energy-efficient design. In *ETS*, pages 1–6, 2013.

[2] G.E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114, 1965.

[3] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Toward dark silicon in servers. *IEEE Micro*, 31(4):6–15, 2011.

[4] H. Esmaeilzadeh, E. Blem, R.S. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *ISCA*, pages 365–376, 2011.

[5] M. Shafique, S. Garg, J. Henkel, and D. Marculescu. The EDA challenges in the dark silicon era: Temperature, reliability, and variability perspectives. In *DAC*, pages 1–6, 2014.

[6] A. Alaghi, W. Qian, and J.P. Hayes. The promise and challenge of stochastic computing. *TCAD*, 37(8):1515–1531, 2018.

[7] S. Venkataramani, S.T. Chakradhar, K. Roy, and A. Raghunathan. Approximate computing and the quest for computing efficiency. In *DAC*, page 120, 2015.

[8] P. Rabinowitz. Multiple-precision division. *Communications of the ACM*, 4(2):98, 1961.

[9] R.E. Goldschmidt. *Applications of division by convergence*. PhD thesis, Massachusetts Institute of Technology, 1964.

[10] M.J. Flynn. On division by functional iteration. *TC*, 100(8):702–706, 1970.

[11] J.N. Mitchell. Computer multiplication and division using binary logarithms. *IRE Transactions on Electronic Computers*, (4):512–517, 1962.

[12] Y.C. Lim. Single-precision multiplier with reduced circuit complexity for signal processing applications. *TC*, 41(10):1333–1336, 1992.

[13] M.J. Schulte and E.E. Swartzlander. Truncated multiplication with correction constant. In *IEEE Workshop on VLSI Signal Processing*, pages 388–396, 1993.

[14] S. Lu. Speeding up processing with approximation circuits. *Computer*, 37(3):67–73, March 2004.

[15] A.W. Burks, H.H. Goldstine, and J. von Neumann. *Preliminary discussion of the logical design of an electronic computing instrument*. Springer, 1947.

[16] A.K. Verma, P. Brisk, and P. Ienne. Variable latency speculative addition: a new paradigm for arithmetic circuit design. In *DATE*, pages 1250–1255, 2008.

[17] N. Zhu, W.L. Goh, and K.S. Yeo. An enhanced low-power high-speed adder for error-tolerant application. In *ISIC*, pages 69–72, 2009.

[18] H.R. Mahdiani, A. Ahmadi, S.M. Fakhraie, and C. Lucas. Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications. *TCAS I*, 57(4):850–862, 2010.

[19] D. Mohapatra, V.K. Chippa, A. Raghunathan, and K. Roy. Design of voltage-scalable meta-functions for approximate computing. In *DATE*, pages 1–6, 2011.

[20] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy. Low-power digital signal processing using approximate adders. *TCAD*, 32(1):124–137, January 2013.

[21] K.Y. Kyaw, W.L. Goh, and K.S. Yeo. Low-power high-speed multiplier for error-tolerant application. In *EDSSC*, pages 1–4, 2010.

[22] P. Kulkarni, P. Gupta, and M. Ercegovac. Trading accuracy for power with an underdesigned multiplier architecture. In *VLSID*, pages 346–351, 2011.

[23] D. Shin. Approximate logic synthesis for error tolerant applications. In *DATE*, pages 957–960, 2010.

[24] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. SALSA: systematic logic synthesis of approximate circuits. In *DAC*, pages 796–801, 2012.

[25] Z. Vasicek and L. Sekanina. Evolutionary approach to approximate digital circuits design. *TEVC*, 19(3):432–444, 2015.

[26] V. Mrazek, S.S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy. Design of power-efficient approximate multipliers for approximate artificial neural networks. In *ICCAD*, page 7, 2016.

[27] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Architecture support for disciplined approximate programming. In *ACM Sigplan Notices*, volume 47, pages 301–312, 2012.

[28] J.S. Miguel, J. Albericio, A. Moshovos, and N.E. Jerger. Doppelgänger: a cache for approximate computing. In *MICRO*, pages 50–61, 2015.

[29] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. EnerJ: Approximate data types for safe and general low-power computation. *ACM Sigplan Notices*, 46(6):164–174, June 2011.

[30] J.Y. L. Low and C.C Jong. Non-iterative high speed division computation based on Mitchell logarithmic method. In *ISCAS*, pages 2219–2222, 2013.

[31] L. Chen, J. Han, W. Liu, and F. Lombardi. Design of approximate unsigned integer non-restoring divider for inexact computing. In *GLSVLSI*, pages 51–56, 2015.

[32] H. Jiang, L. Liu, F. Lombardi, and J. Han. Adaptive approximation in arithmetic circuits: A low-power unsigned divider design. In *DATE*, pages 1411–1416, 2018.

[33] H. Saadat, H. Javaid, and S. Parameswaran. Approximate integer and floating-point dividers with near-zero error bias. In *DAC*, page 161, 2019.

[34] D. Esposito, A.G.M. Strollo, and M. Alioto. Low-power approximate mac unit. In *PRIME*, pages 81–84, 2017.

[35] M.H. Sheu and S.H. Lin. Fast compensative design approach for the approximate squaring function. *JSSC*, 37(1):95–97, 2002.

[36] Y. Chen. Area-efficient fixed-width squarer with dynamic error-compensation circuit. *TCAS II*, 62(9):851–855, 2015.

[37] M.S. Ansari, B.F. Cockburn, and J. Han. Low-power approximate logarithmic squaring circuit design for DSP applications. *TETC*, pages 1–7, 2020.

[38] K.M. Reddy, M.H. Vasantha, Y.N. Kumar, and D. Dwivedi. Design of approximate booth squarer for error-tolerant computing. *TVLSI*, pages 1230–1241, 2020.

[39] H. Jiang, L. Liu, F. Lombardi, and J. Han. Low-power unsigned divider and square root circuit designs using adaptive approximation. *TC*, 68(11):1635–1646, 2019.

[40] L. Chen, J. Han, W. Liu, and F. Lombardi. Algorithm and design of a fully parallel approximate coordinate rotation digital computer (CORDIC). *TMSCS*, 3(3):139–151, 2017.

[41] V.K. Chippa, H. Jayakumar, D. Mohapatra, K. Roy, and A. Raghunathan. Energy-efficient recognition and mining processor using scalable effort design. In *CICC*, 2013.

[42] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM Sigplan Notices*, volume 49, pages 269–284, 2014.

[43] F.S. Snigdha, D. Sengupta, J. Hu, and S.S. Sapatnekar. Optimal design of jpeg hardware under the approximate computing paradigm. In *DAC*, page 106, 2016.

[44] H.A.F. Almurib, T.N. Kumar, and F. Lombardi. Approximate DCT image compression using inexact computing. *TC*, 67(2):149–159, 2017.

[45] M. Brandalero, A.C.S. Beck, L. Carro, and M. Shafique. Approximate on-the-fly coarse-grained reconfigurable acceleration for general-purpose applications. In *DAC*, pages 1–6, 2018.

[46] O. Akbari, M. Kamal, A. Afzali-Kusha, M. Pedram, and M. Shafique. Px-cgra: Polymorphic approximate coarse-grained reconfigurable architecture. In *DATE*, pages 413–418, 2018.

[47] M.S. Ansari, V. Mrazek, B.F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han. Improving the accuracy and hardware efficiency of neural networks using approximate multipliers. *TVLSI*, 28(2):317–328, 2020.

[48] B.J. Yoo, D.H. Lim, and H. Pang, et al. 6.4 A 56 Gb/s 7.7 mW/Gb/s PAM-4 wireline transceiver in 10 nm FinFET using MM-CDR-based ADC timing skew control and low-power DSP with approximate multiplier. In *ISSCC*, pages 122–124, 2020.

[49] C. Liu, J. Han, and F. Lombardi. An analytical framework for evaluating the error characteristics of approximate adders. *TC*, 64(5):1268–1281, 2015.

[50] H. Jiang, C. Liu, F. Lombardi, and J. Han. Low-power approximate unsigned multipliers with configurable error recovery. *TCAS I*, 66(1):189–202, 2018.

[51] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han. A review, classification, and comparative evaluation of approximate arithmetic circuits. *JETC*, 13(4):60, 2017.

[52] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu. ApproxANN: an approximate computing framework for artificial neural network. In *DATE*, pages 701–706, 2015.

[53] R.W. Allred. Circuits, systems, and methods implementing approximations for logarithm, inverse logarithm, and reciprocal, 2007. US Patent 7,171,435.

[54] S.R. Datla, M.A. Thornton, and D.W. Matula. A low power high performance radix-4 approximate squaring circuit. In *ASAP*, pages 91–97, July 2009.

[55] J. Han. Introduction to approximate computing. In *VTS*, pages 1–1, 2016.

[56] W. Liu, F. Lombardi, and M. Shulte. A retrospective and prospective view of approximate computing [Point of View]. *Proceedings of the IEEE*, 108(3):394–399, 2020.

[57] H. Jiang, F.J.H. Santiago, M.S. Ansari, L. Liu, B.F. Cockburn, F. Lombardi, and J. Han. Characterizing approximate adders and multipliers optimized under different design constraints. In *GLSVLSI*, pages 393–398, 2019.

[58] Y. Liu, T. Zhang, and K.K. Parhi. Computation error analysis in digital signal processing systems with overscaled supply voltage. *TVLSI*, 18(4):517–526, 2010.

[59] D. Mohapatra, V.K. Chippa, A. Raghunathan, and K. Roy. Design of voltage-scalable meta-functions for approximate computing. In *DATE*, pages 1–6, 2011.

[60] J. Chen and J. Hu. Energy-efficient digital signal processing via voltage-overscaling-based residue number system. *TVLSI*, 21(7):1322–1332, 2013.

[61] S. Ghosh and K. Roy. Parameter variation tolerance and error resiliency: New design paradigm for the nanoscale era. *Proceedings of the IEEE*, 98(10):1718–1751, 2010.

[62] V.K. Chippa, D. Mohapatra, K. Roy, S.T. Chakradhar, and A. Raghunathan. Scalable effort hardware design. *TVLSI*, 22(9):2004–2016, 2014.

[63] C. Lin and I. Lin. High accuracy approximate multiplier with error correction. In *ICCD*, pages 33–38, October 2013.

[64] M.S. Ansari, B.F. Cockburn, and J. Han. A hardware-efficient logarithmic multiplier with improved accuracy. In *DATE*, pages 928–931, 2019.

[65] G. Zervakis, S. Xydis, K. Tsoumanis, D. Soudris, and K. Pekmestzi. Hybrid approximate multiplier architectures for improved power-accuracy trade-offs. In *ISLPED*, pages 79–84, 2015.

[66] J. Liang, J. Han, and F. Lombardi. New metrics for the reliability of approximate and probabilistic adders. *TC*, 62(9):1760–1771, 2013.

[67] J. Huang, J. Lach, and G. Robins. A methodology for energy-quality tradeoff using imprecise hardware. In *DAC*, pages 504–509, 2012.

[68] J. Miao, K. He, A. Gerstlauer, and M. Orshansky. Modeling and synthesis of quality-energy optimal approximate adders. In *ICCAD*, pages 728–735, 2012.

[69] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan. MACACO: modeling and analysis of circuits for approximate computing. In *ICCAD*, pages 667–673, 2010.

[70] S. Mazahir, O. Hasan, R. Hafiz, M. Shafique, and J. Henkel. Probabilistic error modeling for approximate adders. *TC*, 66(3):515–530, 2017.

[71] M.K. Ayub, O. Hasan, and M. Shafique. Statistical error analysis for low power approximate adders. In *DAC*, pages 1–6, 2017.

[72] A. Qureshi and O. Hasan. Formal probabilistic analysis of low latency approximate adders. *TCAD*, 2018.

[73] M.A. Hanif, R. Hafiz, O. Hasan, and M. Shafique. PEMACx: a probabilistic error analysis methodology for adders with cascaded approximate units. In *DAC*, pages 1–6, 2020.

[74] Silvaco, Inc. Nangate, open cell library. https://www.silvaco.com/products/nangate/Library_Design_Services/index.html.

[75] W. Zhao and Y. Cao. New generation of predictive technology model for sub-45 nm early design exploration. *T-ED*, 53(11):2816–2823, 2006.

[76] H. Amrouch, B. Khaleghi, and A. Gerstlauer. Towards aging-induced approximations. In *DAC*, pages 1–6, 2017.

[77] G.B. Huang, Q.Y. Zhu, and C.K. Siew. Real-time learning capability of neural networks. *IEEE Transactions on Neural Networks*, 17(4):863–878, 2006.

[78] Synopsys, Inc. *DC Ultra*, 2018.

[79] O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram. Dual-quality 4: 2 compressors for utilizing in dynamic accuracy configurable multipliers. *TVLSI*, 25(4):1352–1361, 2017.

[80] F. Sabetzadeh, M.H. Moaiyeri, and M. Ahmadinejad. A majority-based imprecise multiplier for ultra-efficient approximate image multiplication. *TCAS I*, 66(11):4200–4208, 2019.

[81] P.M. Kogge and H.S. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *TC*, 100(8):786–793, 1973.

[82] R.E. Ladner and M.J. Fischer. Parallel prefix computation. *Journal of the ACM*, 27(4):831–838, 1980.

[83] H. Ling. High-speed binary adder. *IBM Journal of Research and Development*, 25(3):156–166, 1981.

[84] R.P. Brent and H.T. Kung. A regular layout for parallel adders. *TC*, (3):260–264, 1982.

[85] T. Han and D.A. Carlson. Fast area-efficient vlsi adders. In *ARITH*, pages 49–56, 1987.

[86] O.J. Bedrij. Carry-select adder. *IRE Transactions on Electronic Computers*, (3):340–346, 1962.

[87] A. Guyot, B. Hochet, and J.M. Muller. A way to build efficient carry-skip adders. *TC*, (10):1144–1152, 1987.

[88] J. Sklansky. Conditional-sum addition logic. *IRE Transactions on Electronic computers*, (2):226–231, 1960.

[89] A. Tyagi. A reduced-area scheme for carry-select adders. *TC*, 42(10):1163–1170, 1993.

[90] Milos D. Ercegovac and Tomas Lang. *Digital Arithmetic*. Elsevier Science & Technology, 2003.

[91] A.B. Kahng and S. Kang. Accuracy-configurable adder for approximate arithmetic designs. In *DAC*, pages 820–825, 2012.

[92] X. Yang, Y. Xing, F. Qiao, Q. Wei, and H. Yang. Approximate adder with hybrid prediction and error compensation technique. In *ISVLSI*, pages 373–378, 2016.

[93] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel. A low latency generic accuracy configurable adder. In *DAC*, pages 1–6, 2015.

[94] M.A. Hanif, R. Hafiz, O. Hasan, and M. Shafique. QuAd: design and analysis of quality-area optimal low-latency approximate adders. In *DAC*, page 42, 2017.

[95] K. Du, P. Varman, and K. Mohanram. High performance reliable variable latency carry select addition. In *DATE*, pages 1257–1262, 2012.

[96] Y. Kim, Y. Zhang, and P. Li. An energy efficient approximate adder with carry skip for error resilient neuromorphic vlsi systems. In *ICCAD*, pages 130–137, 2013.

[97] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu. On reconfiguration-oriented approximate adder design and its application. In *ICCAD*, pages 48–54, 2013.

[98] I. Lin, Y. Yang, and C. Lin. High-performance low-power carry speculative addition with varible latency. *TVLSI*, 23(9):1591–1603, 2015.

[99] L. Li and H. Zhou. On error modeling and analysis of approximate adders. In *ICCAD*, pages 511–518, 2014.

[100] J. Hu and W. Qian. A new approximate adder with low relative error and correct sign calculation. In *DATE*, pages 1449–1454, 2015.

[101] V. Camus, J. Schlachter, and C. Enz. Energy-efficient inexact speculative adder with high performance and accuracy control. In *ISCAS*, pages 45–48, 2015.

[102] V. Camus, J. Schlachter, and C. Enz. A low-power carry cut-back approximate adder with fixed-point implementation and floating-point precision. In *DAC*, page 127, 2016.

[103] F. Ebrahimi-Azandaryani, O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram. Block-based carry speculative approximate adder for energy-efficient applications. *TCAS II*, 2019.

[104] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi. Approximate XOR/XNOR-based adders for inexact computing. In *IEEE-NANO*, pages 690–693, 2013.

[105] H.A.F. Almurib, T.N. Kumar, and F. Lombardi. Inexact designs for approximate low power addition by cell replacement. In *DATE*, pages 660–665, 2016.

[106] M. Pashaeifar, M. Kamal, A. Afzali-Kusha, and M. Pedram. Approximate reverse carry propagate adder for energy-efficient DSP applications. *TVLSI*, 26(11):2530–2541, 2018.

[107] H. Cai, Y. Wang, L.A.B. Naviner, Z. Wang, and W. Zhao. Approximate computing in MOS/spintronic non-volatile full-adder. In *NANOARCH*, pages 203–208, 2016.

[108] S. Angizi, H. Jiang, R.F. DeMara, J. Han, and D. Fan. Majority-based spin-CMOS primitives for approximate computing. *TNANO*, 17(4):795–806, 2018.

[109] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina. EvoApprox8b: library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In *DATE*, pages 258–261, 2017.

[110] Behrooz P. *Computer Arithmetic: Algorithms and Hardware Designs, 2nd edition*. Oxford University Press, New York, 2010.

[111] C.S. Wallace. A suggestion for a fast multiplier. *IEEE Transactions on electronic Computers*, (1):14–17, 1964.

[112] L. Dadda. Some schemes for parallel multipliers. *Alta Frequenza*, 34:349–356, 1965.

[113] N.H.E. Weste and D. Harris. *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.

[114] A.D. Booth. A signed binary multiplication technique. *The Quarterly Journal of Mechanics and Applied Mathematics*, 4(2):236–240, 1951.

[115] O.L. MacSorley. High-speed arithmetic in binary computers. *Proceedings of the IRE*, 49(1):67–91, 1961.

[116] C.R. Baugh and B.A. Wooley. A two's complement parallel array multiplication algorithm. *TC*, 100(12):1045–1047, 1973.

[117] M. Hatamian and G.L. Cash. A 70-MHz 8-bit×8-bit parallel pipelined multiplier in 2.5-μm CMOS. *JSSC*, 21(4):505–513, 1986.

[118] J. Wang, S. Kuang, and S. Liang. High-accuracy fixed-width modified Booth multipliers for lossy applications. *TVLSI*, 19(1):52–60, 2011.

[119] K. Bhardwaj, Pravin S. Mane, and J. Henkel. Power- and area-efficient approximate Wallace tree multiplier for error-resilient systems. In *ISQED*, pages 263–269, 2014.

[120] S. Hashemi, R. Bahar, and S. Reda. DRUM: a dynamic range unbiased multiplier for approximate applications. In *ICCAD*, pages 418–425, 2015.

[121] C. Liu, J. Han, and F. Lombardi. A low-power, high-performance approximate multiplier with configurable partial error recovery. In *DATE*, pages 1–4, 2014.

[122] J. Ma, K.L. Man, N. Zhang, S. Guan, and T.T. Jeong. High-speed area-efficient and power-aware multiplier design using approximate compressors along with bottom-up tree topology. In *ICMV: Algorithms, Pattern Recognition, and Basic Technologies*, volume 8784, page 87841Z, 2013.

[123] A. Momeni, J. Han, P. Montuschi, and F. Lombardi. Design and analysis of approximate compressors for multiplication. *TC*, 64(4):984–994, 2015.

[124] S. Venkatachalam and S.B. Ko. Design of power and area efficient approximate multipliers. *TVLSI*, 25(5):1782–1786, 2017.

[125] M.S. Ansari, H. Jiang, B.F. Cockburn, and J. Han. Low-power approximate multipliers using encoded partial products and approximate compressors. *JETC*, 8(3):404–416, 2018.

[126] D. Esposito, A.G.M. Strollo, E. Napoli, D. De Caro, and N. Petra. Approximate multipliers based on new approximate compressors. *TCAS I*, 65(12):4169–4182, 2018.

[127] W. Liu, J. Xu, D. Wang, C. Wang, P. Montuschi, and F. Lombardi. Design and evaluation of approximate logarithmic multipliers for low power error-tolerant applications. *TCAS I*, 65(9):2856–2868, 2018.

[128] M.S. Ansari, B.F. Cockburn, and J. Han. An improved logarithmic multiplier for energy-efficient neural computing. *TC*, pages 1–12, 2020.

[129] H. Saadat, H. Javaid, A. Ignjatovic, and S. Parameswaran. REALM: reduced-error approximate log-based integer multiplier. In *DATE*, pages 1366–1371, 2020.

[130] H. Kim, M.S. Kim, A.A. Del Barrio, and N. Bagherzadeh. A cost-efficient iterative truncated logarithmic multiplication for convolutional neural networks. In *ARITH*, pages 108–111, 2019.

[131] V. Mrazek, Z. Vasicek, L. Sekanina, H. Jiang, and J. Han. Scalable construction of approximate multipliers with formally guaranteed worst case error. *TVLSI*, (99):2572–2576, 2018.

[132] G. Zervakis, K. Tsoumanis, S. Xydis, D. Soudris, and K. Pekmestzi. Design-efficient approximate multiplication circuits through partial product perforation. *TVLSI*, 24(10):3105–3117, 2016.

[133] S. Narayanamoorthy, H.A. Moghaddam, Z. Liu, T. Park, and N.S. Kim. Energy-efficient approximate multiplication for digital signal processing and classification applications. *TVLSI*, 23(6):1180–1184, 2015.

[134] D. Kelly, B. Phillips, and S. Al-Sarawi. Approximate signed binary integer multipliers for arithmetic data value speculation. In *ECSI*, 2009.

[135] C. Liu. Design and analysis of approximate adders and multipliers. Master's thesis, University of Alberta, Canada, 2014.

[136] K. Cho, K. Lee, J. Chung, and K.K. Parhi. Design of low-error fixed-width modified Booth multiplier. *TVLSI*, 12(5):522–531, 2004.

[137] M.A. Song, L.D. Van, and S.Y. Kuo. Adaptive low-error fixed-width Booth multipliers. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 90(6):1180–1187, 2007.

[138] F. Farshchi, M.S. Abrishami, and S.M. Fakhraie. New approximate multiplier for low power digital signal processing. In *CADS*, pages 25–30, October 2013.

[139] Y. Chen and T. Chang. A high-accuracy adaptive conditional-probability estimator for fixed-width Booth multipliers. *TCAS I*, 59(3):594–603, 2012.

[140] W. Liu, L. Qian, C. Wang, H. Jiang, J. Han, and F. Lombardi. Design of approximate radix-4 Booth multipliers for error-tolerant computing. *TC*, 66(8):1435–1441, 2017.

[141] B. Shao and P. Li. Array-based approximate arithmetic computing: a general model and applications to multiplier and squarer design. *TCAS I*, 62(4):1081–1090, 2015.

[142] H. Jiang, J. Han, F. Qiao, and F. Lombardi. Approximate radix-8 Booth multipliers for low-power and high-performance operation. *TC*, 65(8):2638–2644, 2016.

[143] S.F. Oberman and M.J. Flynn. Design issues in division and other floating-point operations. *TC*, 46(2):154–161.

[144] M. Cappa and V.C. Hamacher. An augmented iterative array for high-speed binary division. *TC*, C-22(2):172–175, 1973.

[145] D.W. Sweeney. Divider device for skipping a string of zeros or radix-minus-one digits, 1964. US Patent 3,145,296.

[146] J.E. Robertson. A new class of digital division methods. *IRE Transactions on Electronic Computers*, (3):218–222, 1958.

[147] K.D. Tocher. Techniques of multiplication and division for automatic binary computers. *The Quarterly Journal of Mechanics and Applied Mathematics*, 11(3):364–384, 1958.

[148] W. Liu and A. Nannarelli. Power efficient division and square root unit. *TC*, 61(8):1059–1070, 2012.

[149] M.D. Ercegovac and T. Lang. On-the-fly conversion of redundant into conventional representations. *TC*, C-36(7):895–897, 1987.

[150] L. Chen, J. Han, W. Liu, and F. Lombardi. On the design of approximate restoring dividers for error-tolerant applications. *TC*, 65(8):2522–2533, 2016.

[151] L. Chen, F. Lombardi, P. Montuschi, J. Han, and W. Liu. Design of approximate high-radix dividers by inexact binary signed-digit addition. In *GLSVLSI*, pages 293–298, 2017.

[152] L. Chen, J. Han, W. Liu, P. Montuschi, and F. Lombardi. Design, evaluation and application of approximate high-radix dividers. *TMSCS*, 4(3):299–312, 2018.

[153] S. Hashemi, R. Bahar, and S. Reda. A low-power dynamic divider for approximate applications. In *DAC*, page 105, 2016.

[154] R. Zendegani, et al. SEERAD: a high speed yet energy-efficient rounding-based approximate divider. In *DATE*, pages 1481–1484, 2016.

[155] W. Liu, J. Li, T. Xu, C. Wang, P. Montuschi, and F. Lombardi. Combining restoring array and logarithmic dividers into an approximate hybrid design. In *ARITH*, pages 92–98, 2018.

[156] M. Imani, R. Garcia, A. Huang, and T. Rosing. CADE: configurable approximate divider for energy efficiency. In *DATE*, pages 586–589, 2019.

[157] L. Wu and C.C. Jong. A curve fitting approach for non-iterative divider design with accuracy and performance trade-off. In *NEWCAS*, pages 1–4, 2015.

[158] M.S.K. Lau, K. Ling, and Y. Chu. Energy-aware probabilistic multiplier: design and analysis. In *CASES*, pages 281–290, 2009.

[159] R.C. Gonzalez and R.E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., 2007.

[160] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, 2016.

[161] H. Mo, L. Liu, W. Zhu, Q. Li, H. Liu, W. Hu, Y. Wang, and S. Wei. A 1.17 TOPS/W, 150 fps accelerator for multi-face detection and alignment. In *DAC*, page 80, 2019.

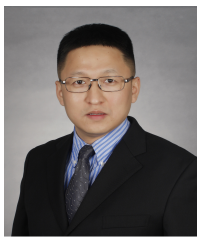
[162] V. Jain and E. Learned-Miller. FDDB: A benchmark for face detection in unconstrained settings. Technical Report UM-CS-2010-009, University of Massachusetts, Amherst, 2010.

[163] M. Koestinger, P. Wohlhart, P.M. Roth, and H. Bischof. Annotated facial landmarks in the wild: a large-scale, real-world database for facial landmark localization. In *ICCV workshops*, pages 2144–2151, 2011.

Honglan Jiang (S'14-M'18) received the B.Sc. and Master degrees in instrument science and technology from the Harbin Institute of Technology, Harbin, Heilongjiang, China, in 2011 and 2013, respectively. In 2018, she received the Ph.D. degree in integrated circuits and systems from the University of Alberta, Edmonton, AB, Canada. She is currently a postdoctoral fellow in the Institute of Microelectronics, Tsinghua University, Beijing, China. Her research interests include approximate computing, reconfigurable computing and stochastic computing.

Francisco Javier Hernandez Santiago received the B.Eng. degree in electronics and communications from the University of Guadalajara, Guadalajara, Mexico, in 2012, and the M.Sc. degree in computer engineering from the University of Alberta, Edmonton, Canada, in 2020. He is currently a security engineer at Intel, Zapopan, Mexico. His research interests include approximate computing, computer architecture, and computer security.

Hai Mo received B.S. degree in computer science from Huazhong University Of Science And Technology, Wuhan, Hubei, China in 2019. He is currently working toward the M.S. degree in the Institute of Microelectronics at Tsinghua University, Beijing, China, on approximation algorithm and high-performance computing. His research interest spans the area of architecture design, in-memory computing, and hardware implementation for deep learning accelerators.



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

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