Low-Power Approximate Unsigned Multipliers With Configurable Error Recovery

Honglan Jiang, *Student Member, IEEE*, Cong Liu, Fabrizio Lombardi, *Fellow, IEEE*, and Jie Han, *Senior Member, IEEE*

***Abstract*— Approximate circuits have been considered for applications that can tolerate some loss of accuracy with improved performance and/or energy efficiency. Multipliers are key arithmetic circuits in many of these applications including digital signal processing (DSP). In this paper, a novel approximate multiplier with a low power consumption and a short critical path is proposed for high-performance DSP applications. This multiplier leverages a newly designed approximate adder that limits its carry propagation to the nearest neighbors for fast partial product accumulation. Different levels of accuracy can be achieved by using either OR gates or the proposed approximate adder in a configurable error recovery circuit. The approximate multipliers using these two error reduction strategies are referred to as AM1 and AM2, respectively. Both AM1 and AM2 have a low mean error distance, i.e., most of the errors are not significant in magnitude. Compared with a Wallace multiplier optimized for speed, an 8 × 8 AM1 using four most significant bits for error reduction shows a 60% reduction in delay (when optimized for delay) and a 42% reduction in power dissipation (when optimized for area). In a 16 × 16 design, half of the least significant partial products are truncated for AM1 and AM2, which are thus denoted as TAM1 and TAM2, respectively. Compared with the Wallace multiplier, TAM1 and TAM2 save from 50% to 66% in power, when optimized for area. Compared with existing approximate multipliers, AM1, AM2, TAM1, and TAM2 show significant advantages in accuracy with a low power-delay product. AM2 has a better accuracy compared with AM1 but with a longer delay and higher power consumption. Image processing applications, including image sharpening and smoothing, are considered to show the quality of the approximate multipliers in error-tolerant applications. By utilizing an appropriate error recovery scheme, the proposed approximate multipliers achieve similar processing accuracy as exact multipliers, but with significant improvements in power.**

***Index Terms*— Approximate computing, multiplier, adder, error recovery, low-power, image processing.**

Manuscript received July 7, 2017; revised January 20, 2018 and March 15, 2018; accepted July 3, 2018. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada under Project RES0025211. This paper was recommended by Associate Editor Y. Ha. *(Honglan Jiang and Cong Liu contributed equally to this work.) (Corresponding author: Jie Han.)*

H. Jiang, C. Liu, and J. Han are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada (e-mail: honglan@ualberta.ca; cong4@ualberta.ca; jhan8@ualberta.ca).

F. Lombardi is with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115 USA (e-mail: lombardi@ece.neu.edu).

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

Digital Object Identifier 10.1109/TCSI.2018.2856245

I. Introduction

APPROXIMATE computing has emerged as a potential solution for the design of energy-efficient digital systems [1]. Applications such as multimedia, recognition and data mining are inherently error-tolerant and do not require a perfect accuracy in computation. For digital signal processing (DSP) applications, the result is often left to interpretation by human perception. Therefore, strict exactness may not be required and an imprecise result may suffice due to the limitation of human perception. For these applications, approximate circuits play an important role as a promising alternative for reducing area, power and delay, thereby achieving better performance in energy efficiency.

As one of the key components in arithmetic circuits, adders have been extensively studied for approximate implementation [2]–[8]. As the typical carry propagation chain is usually shorter than the width of an adder, the speculative adders use a reduced number of less significant input bits to calculate the sum bits [2]. An error detection and recovery scheme has been proposed in [3] to extend the scheme of [2] for a reliable adder with variable latency. A reliable variable-latency adder based on carry select addition has been presented in [8]. As a number of approximate adders have been proposed, new methodologies to model, analyze and evaluate them have been discussed in [9]–[12].

A multiplier usually consists of three stages: partial product generation, partial product accumulation and a carry propagation adder (CPA) at the final stage [13]. In the underdesigned multiplier (UDM), approximate partial products are computed using inaccurate 2×2 multiplier blocks, while accurate adders are used in an adder tree to accumulate the approximate partial products [14]. In [15], approximate 4×4 and 8×8 bit Wallace multipliers are designed by using a carry-in prediction method. Then, they are used in the design of approximate 16×16 Wallace multipliers, referred to as AWTM. The AWTM is configured into four different modes by using a different number of approximate 4×4 and 8×8 multipliers. The use of approximate speculative adders has been discussed in [10] for the final stage addition in a multiplier. The error tolerant multiplier (ETM) of [16] is based on the partition of a multiplier into an accurate multiplication part for most significant bits (MSBs) and a non-multiplication part for least significant bits (LSBs). The static segment multiplier (SSM) utilizes a similar partition scheme [17]. In an $n \times n$ SSM, an $m \times m$ accurate multiplier

($m \geqslant n/2$) is used to multiply the m consecutive bits from the two input operands. Whether the $(n-m)$ MSBs of each input operand are all zero determines the selection of the inputs for the accurate multiplier (m MSBs or m LSBs). These approximate multipliers are designed for unsigned operation. Signed multiplication is usually implemented by using a Booth algorithm. Approximate designs have been proposed for fixed-width Booth multipliers [18]–[20].

In this paper, a novel approximate multiplier design is proposed using a simple, yet fast approximate adder. This newly designed adder can process data in parallel by cutting the carry propagation chain. It has a critical path delay that is shorter than a conventional one-bit full adder. Albeit with a high error rate, this adder simultaneously computes the sum and generates an error signal; this feature is employed to reduce the error in the final result of the multiplier. In the proposed approximate multiplier, a simple tree of the approximate adders is used for partial product accumulation and the error signals are used to compensate errors for obtaining a better accuracy.

The proposed multiplier can be configured into two designs by using OR gates and the proposed approximate adders for error reduction, referred to as approximate multiplier 1 (AM1) and approximate multiplier 2 (AM2), respectively. Different levels of error recovery can also be achieved by using a different number of MSBs for error recovery in both AM1 and AM2. As per the analysis, the proposed multipliers have significantly shorter critical paths and lower power dissipation than the traditional Wallace multiplier. Functional and circuit simulations are performed to evaluate the performance of the multipliers. Image sharpening and smoothing are considered as approximate multiplication-based DSP applications. Experimental results indicate that the proposed approximate multipliers perform well in these error-tolerant applications. The proposed designs can be used as effective library cells for the synthesis of approximate circuits [21], [22].

This paper is a significant extension of [23] and is organized as follows. Section II presents the proposed approximate adder and the design of the multiplier. Section III discusses the error reduction schemes for AM1 and AM2. Section IV shows the accuracy results and in section V, delay, area and power consumption are obtained. Section VI compares the proposed approximate multipliers with the existing designs in terms of accuracy and hardware costs. Section VII discusses the application of the proposed multiplier to image processing applications. Section VIII concludes the paper.

II. Proposed Approximate Multiplier

A. The Approximate Adder

In this section, the design of a new approximate adder is presented. This adder operates on a set of pre-processed inputs. The input pre-processing (IPP) is based on the *commutativity* of bits with the same weights in different addends. For example, consider two sets of inputs to a 4-bit adder: i) $A = 1010, B = 0101$ and ii) $A = 1111, B = 0000$. Clearly, the additions in i) and ii) produce the same result. In this process, the two input bits $A_iB_i = 01$ are equivalent to $A_iB_i = 10$ (with i being the bit index) due to the commutativity of the corresponding bits in the two operands.

TABLE I

Truth Table of the Approximate Adder Cell. 'X' Represents That no Such a Combination Occurs Due to the IPP

S_i/E_i		$\dot{B}_i\dot{B}_{i-1}$ 00	01	11	10
$\dot{A}_i\dot{A}_{i-1}$	00	0/0	X	X	X
	01	0/0	1/0	X	X
	11	1/0	1/1	1/0	0/0
	10	1/0	X	X	0/0

The basic rule for the IPP is to switch A_i and B_i if $A_i = 0$ and $B_i = 1$ (for any i), while keeping the other combinations (i.e., $A_iB_i = 00, 10$ and 11) unchanged. By doing so, more 1's are expected in A and more 0's are expected in B. If $\dot{A}_i\dot{B}_i$ are the i^{th} bits in the pre-processed inputs, the IPP functions are given by:

$$\dot{A}_i = A_i + B_i, \tag{1}$$
$$\dot{B}_i = A_iB_i. \tag{2}$$

Equations (1) and (2) compute the propagate and generate signals used in a parallel adder such as the carry look-ahead adder (CLA). The proposed adder can process data in parallel by cutting the carry propagation chain. Let A and B denote the two input binary operands of an adder, S be the sum result, and E represent the error vector. A_i, B_i, S_i and E_i are the i^{th} least significant bits of A, B, S and E, respectively. A carry propagation chain starts at the i^{th} bit when $\dot{B}_i = 1$, $\dot{A}_{i+1} = 1$, $\dot{B}_{i+1} = 0$. In an accurate adder, S_{i+1} is 0 and the carry propagates to the higher bit. However, in the proposed approximate adder, S_{i+1} is set to 1 and an error signal is generated as $E_{i+1} = 1$. This prevents the carry signal from propagating to the higher bits. Hence, a carry signal is produced only by the generate signal, i.e., $C_i = 1$ only when $\dot{B}_i = 1$, and it only propagates to the next higher bit, i.e., the $(i+1)^{th}$ position. Table I shows the truth table of the approximate adder, where $\dot{A}_i$, $\dot{B}_i$ and $\dot{B}_{i-1}$ are the inputs after IPP. The error signal is utilized for error compensation purposes as discussed in a later section. In this case, the approximate adder is similar to a redundant number system [24] and the logical functions of Table I are given by

$$S_i = \dot{B}_{i-1} + \overline{\dot{B}_i}\dot{A}_i, \tag{3}$$
$$E_i = \overline{\dot{B}_i}\dot{B}_{i-1}\dot{A}_i. \tag{4}$$

By replacing $\dot{A}_i$ and $\dot{B}_i$ using (1) and (2) respectively, the logic functions with respect to the original inputs are given by

$$S_i = (A_i \oplus B_i) + A_{i-1}B_{i-1}, \tag{5}$$
$$E_i = (A_i \oplus B_i)A_{i-1}B_{i-1}, \tag{6}$$

where i is the bit index, i.e., $i = 0, 1, \cdots, n$ for an n-bit adder. Let $A_{-1} = B_{-1} = 0$ when i is 0, thus, $S_0 = A_0 \oplus B_0$ and $E_0 = 0$. Also, $E_i = 0$ when A_{i-1} or B_{i-1} is 0.

Consider an n-bit adder, the inputs are given by $A = A_{n-1} \cdots A_1A_0$ and $B = B_{n-1} \cdots B_1B_0$, the exact sum is

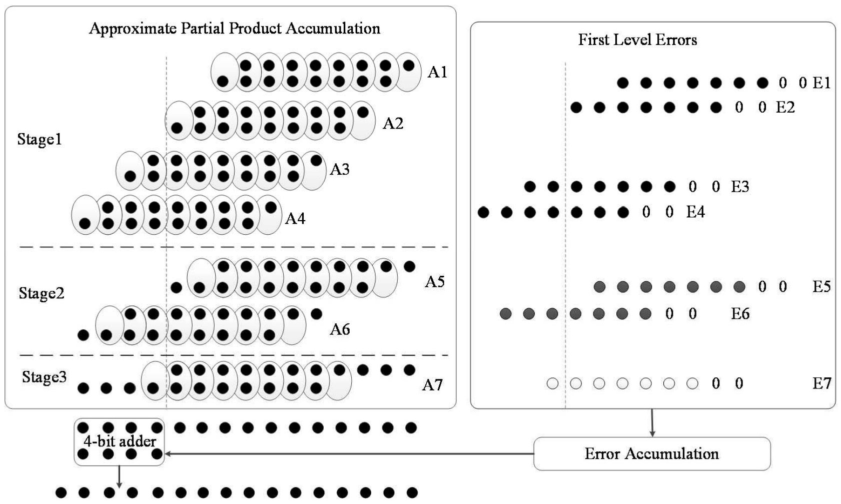

Fig. 1. An approximate multiplier with partial error recovery using 5 MSBs of the error vector. ●: a partial product, sum or an error bit generated at the first stage; ◉: an error bit generated at the second stage; ○: an error bit generated at the last stage.

$\widetilde{S} = \widetilde{S}_{n-1} \cdots \widetilde{S}_1 \widetilde{S}_0$. Then, $\widetilde{S}_i$ can be computed as $S_i + E_i$ and thus, the exact sum of A and B is given by

$$\widetilde{S} = S + E. \tag{7}$$

In (7) '+' means the addition of two binary numbers rather than the 'OR' function. The error E is always non-negative and the approximate sum is always equal to or smaller than the accurate sum. This is an important feature of this adder because an additional adder can be used to add the error to the approximate sum as a compensation step. While this is intuitive in an adder design, it is a particularly useful feature in a multiplier design as only one additional adder is needed to reduce the error in the final product.

B. Proposed Approximate Multiplier

A distinguishing feature of the proposed approximate multiplier is the simplicity to use approximate adders in the partial product accumulation. It has been shown that this may lead to low accuracy [14], because errors may accumulate and it is difficult to correct errors using existing approximate adders. However, the use of the newly proposed approximate adder overcomes this problem by utilizing the error signal. The resulting design has a critical path delay that is shorter than a conventional one-bit full adder, because the new n-bit adder can process data in parallel. The approximate adder has a rather high error rate, but the feature of generating both the sum and error signals at the same time reduces errors in the final product. An adder tree is utilized for partial product accumulation; the error signals in the tree are then used to compensate the error in the output to generate a product with a better accuracy.

The architecture of the proposed approximate multiplier is shown in Fig. 1. In the proposed design, the simplification of the partial product accumulation stage is accomplished by using an adder tree, in which the number of partial products is reduced by a factor of 2 at each stage of the tree. This adder tree is usually not implemented using accurate multi-bit adders due to the long latency. However, the proposed approximate

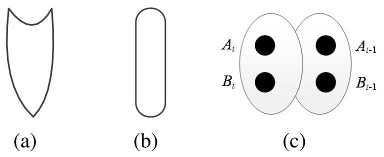

Fig. 2. Symbols for (a) an OR gate, (b) a full adder or a half adder and (c) an approximate adder cell

adder is suitable for implementing an adder tree, because it is less complex than a conventional adder and has a much shorter critical path delay.

Exact fast multipliers often include a Wallace or Dadda tree using full adders (FAs) and half adders (HAs); compressors are also utilized in the Wallace or Dadda tree to further reduce the critical path with an increase in circuit area. These designs require a proper selection of different circuit modules; for example, 4:2 compressors, FAs and HAs are commonly used in a Wallace tree and a judicious connection of these modules must be considered in a tree design. This increases the design complexity, especially when multipliers of different sizes are considered; the proposed design is simple for various multiplier sizes.

III. Error Reduction

The approximate adder generates two signals: the approximate sum S and the error E; the use of the error signal is considered next to reduce the inaccuracy of the multiplier. As (7) is applicable to the sum of every single approximate adder in the tree, an error reduction circuit is applied to the final multiplication result rather than to the output of each adder. Two steps are required to reduce errors: i) error accumulation and ii) error recovery by the addition of the accumulated errors to the adder tree output using a CPA. In the error accumulation step, error signals are accumulated to a single error vector, which is then added to the output vector of the partial product accumulation tree. Two approximate error accumulation methods are proposed, yielding the approximate multiplier 1 (AM1) and approximate multiplier 2 (AM2). Fig. 2 shows the symbols for an OR gate, a full adder and half adder cell and an approximate adder cell used in the error accumulation tree.

A. Error Accumulation for Approximate Multiplier 1

As shown in Fig. 1, each approximate adder Ai generates a sum vector Si and an error vector Ei, where $i = 1, 2, \cdots, 7$. If the error signals are added using accurate adders, the accumulated error can fully compensate the inaccurate product; however to reduce complexity, an approximate error accumulation is introduced. Consider the observation that the error vector of each approximate adder tends to have more 0's than 1's. Therefore, the probability that the error vectors have an error bit '1'at the same position, is quite small. Hence, an OR gate is used to approximately compute the sum of the errors for a single bit. If *m* error vectors (denoted by $E1$, $E2$, ..., Em) have to be accumulated, then the sum of these vectors is obtained as

$$E_i = E1_i \ OR \ E2_i \ OR \ ... \ OR \ Em_i. \tag{8}$$

To reduce errors, an accumulated error vector is added to the adder tree output using a conventional CPA (e.g. a carry look-ahead adder). However, only several (e.g. k) MSBs of the error signals are used to compensate the outputs to further reduce the overall complexity. The number of MSBs is selected according to the extent that errors must be compensated. For example in an 8×8 adder tree, there are a total of 7 error vectors, generated by the 7 approximate adders in the tree. However, not all the bits in the 7 vectors need to be added, because the MSBs of some vectors are less significant than the least significant bits of the k MSBs. In the example of Fig. 1, 5 MSBs (i.e. the $(11-14)^{th}$ bits, no error is generated at the 15^{th} bit position) are considered for error recovery and therefore, 4 error vectors are considered (i.e., the error vectors $E3$, $E4$, $E6$ and $E7$). The error vectors of the other three adders are less significant than the 11^{th} bit, so they are not considered. The accumulated error E is obtained using (8); then, the final result is found by adding E to S using a fast accurate CPA. The error accumulation scheme is shown in Fig. 3. As no error is generated at the least significant two bits of each approximate adder Ai $(i = 1, 2, \cdots, 7)$, the least significant two bits of each error vector Ei are not accumulated.

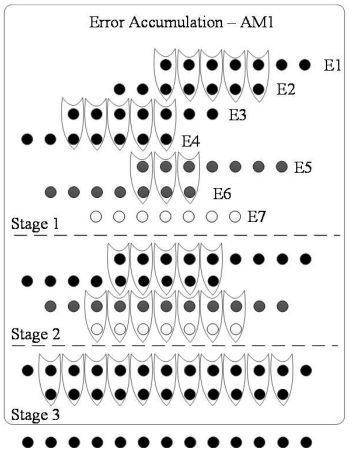

Fig. 3. Error accumulation tree for AM1.

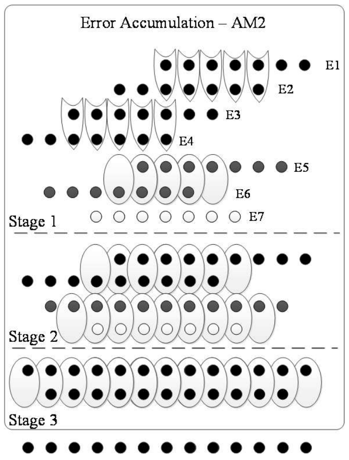

Fig. 4. Error accumulation tree for AM2.

B. Error Accumulation for Approximate Multiplier 2

The error accumulation scheme for AM2 is shown in Fig. 4. To introduce the design of AM2, an 8×8 multiplier with two inputs X and Y is considered. For example, consider the first two partial product vectors $X_0Y_7, X_0Y_6, ..., X_0Y_0$ and $X_1Y_7, X_1Y_6, ..., X_1Y_0$ accumulated by the first approximate adder (A1 in Fig. 1), where X_i and Y_i are the i^{th} least significant bits of X and Y, respectively. Recall from (6) for the approximate adder, the condition for $E_i = 1$ is

$$A_{i-1} = B_{i-1} = 1 \quad \text{and} \quad A_i \neq B_i. \tag{9}$$

For the first approximate adder in the partial product accumulation tree, its inputs are $A = X_0Y_7, X_0Y_6, ..., X_0Y_0$ and $B = X_1Y_7, X_1Y_6, ..., X_1Y_0$. Thus, the i^{th} least significant bits for A and B are $A_i = X_0Y_i$ and $B_i = X_1Y_{i-1}$, respectively. If X_0 or X_1 is 0, there will be no error in this approximate adder because either A or B is zero. Therefore, no error occurs unless $X_0X_1 = 11$. When $X_0X_1 = 11$, A_i and B_i are simplified to Y_i and Y_{i-1}, respectively. Then to calculate E_i, A_{i-1}, B_{i-1}, A_i and B_i are replaced by Y_{i-1}, Y_{i-2}, Y_i and Y_{i-1}, respectively. For E_i to be 1, $Y_iY_{i-1}Y_{i-2} = 011$ according to (9). Therefore, an error only occurs when the input has "011" as a bit sequence. Based on this observation, the "distance" between two errors in an approximate multiplier is at least 3 bits. Thus, two neighboring approximate adders in the first stage of the partial product tree cannot have errors at the same column, because the errors in a lower approximate adder are those in the upper adder shifted by 2 bits when both errors exist. The errors in two neighboring approximate adders can then be accurately accumulated by OR gates, e.g., an OR gate can be used to accumulate the two bits in the error vectors $E1$ and $E2$ in Fig. 1. After applying the OR gates to accumulate $E1$ and $E2$ as well as $E3$ and $E4$, the four error vectors are compressed into two. For $E5$, $E6$ and $E7$, they are generated from the approximate sum of the partial products rather than the partial products. Therefore, they cannot be accurately accumulated by OR gates.

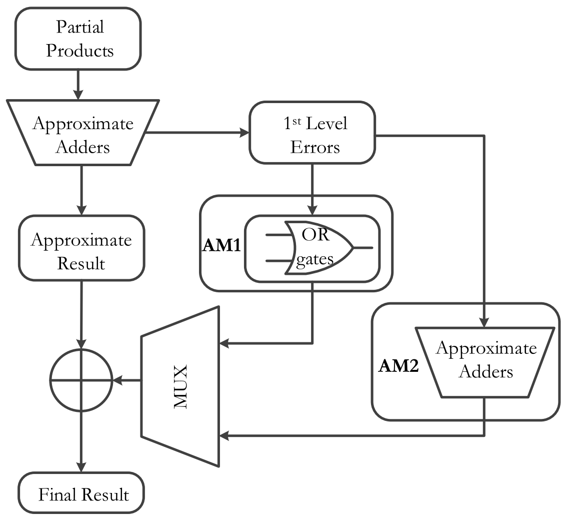

Fig. 5. Block diagram of the proposed multipliers.

Another interesting feature of the proposed approximate adder is as follows. Assume $E_i = 1$ in (6), then $A_{i-1} = B_{i-1} = 1$ and $A_i \neq B_i$. Since $A_{i-1} = B_{i-1} = 1$, i.e., $A_{i-1} \oplus B_{i-1} = 0$, it is easy to show that $E_{i-1} = 0$. Moreover, as $A_i \neq B_i$, i.e., $A_i B_i = 0$, then $E_{i+1} = 0$. Thus, once there is an error in one bit, its neighboring bits are error free, i.e., there are no consecutive error bits in one row. Therefore, there is no carry propagation path longer than two bits when two error vectors are accumulated, and the error vectors are accurately accumulated by the proposed approximate adder.

Based on the above analysis, $E5$ and $E6$ are accurately accumulated by one approximate adder in the first stage of the error accumulation. After the first stage of error accumulation, three vectors are generated, and another two approximate adders are then used to accumulate these three vectors as well as the error vector remaining from the previous stage ($E7$). Simulation results (found in later sections) show that the modified error accumulation outperforms the OR-gate error accumulation with little overhead on delay and power.

Hereafter, the proposed $n \times n$ approximate multiplier with k-MSB OR-gate based error reduction is referred to as an n/k AM1, while an $n \times n$ approximate multiplier with k-MSB approximate adder based error reduction is referred to as an n/k AM2. The structures of AM1 and AM2 are shown in Fig. 5.

C. 16 × 16 *Approximate Multipliers*

In both AM1 and AM2, all the error vectors are compressed to one error vector, which is then added back to the approximate output of the partial product tree. Compared to 8 × 8 designs, 16 × 16 multipliers generate more error vectors, and too much information would be ignored if the same error reduction strategies are used. That is, using only one compressed error vector does not make a good estimate of the overall error. In the modified designs, the error vectors generated by the approximate adders are compressed to two final error vectors. Take a 16 × 16 AM1 as an example, the eight error vectors generated at the first stage of the partial product accumulation tree are compressed to one error vector, EV1, using OR gates. The remaining seven error vectors from the second, third and fourth stages are compressed to another error vector EV2. Then both EV1 and EV2 are added back to the output of the partial product at the fourth stage. Similarly, the proposed approximate adders are used in a 16×16 AM2 to compress the eight error vectors from the first stage to one error vector and the remaining error vectors to another error vector.

Truncation can also be applied to the proposed designs for large input operands. Therefore, 16 LSBs of the partial products are truncated in 16 × 16 AM1 and AM2, resulting in truncated AM1 (TAM1) and truncated AM2 (TAM2).

IV. Accuracy Evaluation

Arithmetic accuracy in approximate circuits is compromised for improvements in other metrics (such as reduced circuit complexity and delay). In [9], the error distance (ED) and mean error distance (MED) are proposed to evaluate the performance of approximate arithmetic circuits. For multipliers, ED is defined to be the arithmetic difference between the accurate product (M) and the approximate product (M'), i.e.,

$$ED = |M' - M|. \tag{10}$$

MED is the average of EDs for a set of outputs (obtained by applying a set of inputs). A metric applicable for comparing multipliers of different sizes is the normalized MED (NMED), i.e.,

$$NMED = \frac{MED}{M_{max}}, \tag{11}$$

where M_{max} is the maximum magnitude of the output of an accurate multiplier, i.e. $(2^n - 1)^2$ for an $n \times n$ multiplier. The relative error distance (RED) is defined as:

$$RED = |\frac{M' - M}{M}| = \frac{ED}{|M|}. \tag{12}$$

Similarly, the mean relative error distance (MRED) can be obtained.

The error rate (ER) is defined as the percentage of erroneous outputs among all outputs [25]. For evaluating the worst-case output, the maximum error (ME) is defined as the maximum error distance normalized by the maximum output of the accurate multiplier. In this paper, the NMED, MRED, ER and ME are used to evaluate the proposed multipliers.

A. Accuracy Evaluation of 8 × 8 *Multipliers*

As an error can occur at any stage (e.g., the partial product accumulation stage and the error accumulation stage) and complicated correlations exist, it is difficult, if not impossible, to develop mathematical models for the error analysis of the approximate multipliers. Thus, the functions of the proposed multipliers are realized using Matlab and an exhaustive simulation is performed for an 8 × 8 approximate multiplier.

Approximate multipliers with both the OR gate and the approximate adder based error reduction, as well as the

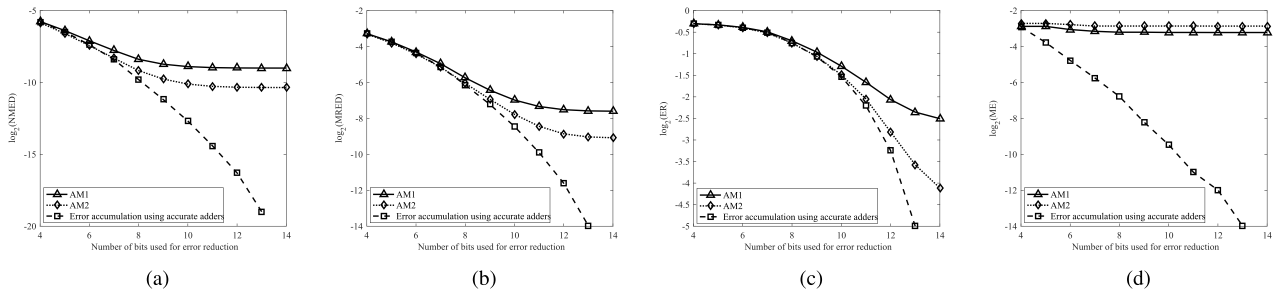

Fig. 6. Accuracy comparison of the approximate 8 × 8 multipliers using approximate and exact error accumulation vs. different number of bits for error reduction. (a) NMED. (b) MRED. (c) ER. (d) ME.

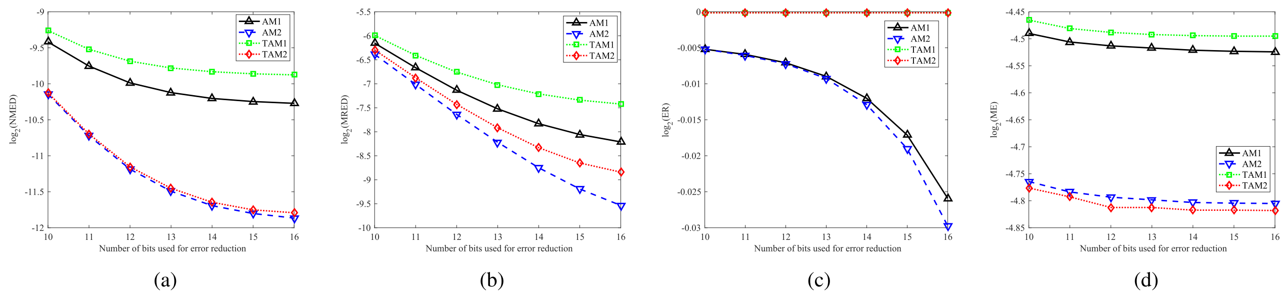

Fig. 7. Accuracy comparison of the approximate 16 × 16 multipliers vs. the number of bits used for error reduction. (a) NMED. (b) MRED. (c) ER. (d) ME.

accurate adder based error reduction, are evaluated. Fig. 6 shows the four metrics (NMED, MRED, ER and ME) in logarithm when using different numbers of MSBs for error reduction. For the approximate multipliers, there is no error in the least significant 2 bits of the output, so the largest number of MSBs used for error reduction is 14. Let k denote the number of MSBs used for error reduction. The values of NMED and MRED of AM1 and AM2 drop drastically as k is increased from 4 to 8 and continues to drop as k increases, even though at a slower rate. In terms of ER, the values for the proposed multipliers decrease slowly with an increasing k from 4 to 8 and then follow a sharper decline. The MEs for AM1 and AM2 do not decrease as much as the multiplier with an accurate error accumulation when k increases. This occurs because some errors at the higher bit positions are not accurately accumulated by using the OR gates or the proposed approximate adders. The values of NMED, MRED, ER and ME finally drop to zero for the accurate error accumulation when 14 MSBs are used for error reduction (not shown in Fig. 6 because the logarithmic values are infinite).

For the same k, AM2 has a better performance than AM1 in terms of NMED, MRED and ER. For example, if 8 MSBs are used for error reduction, the NMED of AM2 is 0.17% while it is 0.30% for AM1. Moreover, if 14 MSBs are used for error reduction, AM1 has an error rate of 17.6%, while the error rate of AM2 can be as low as 5.8%.

These four figures also indicate that the proposed approximate multiplier has a rather high error rate, but the errors are usually very small compared to both the accurate and the largest possible output of the approximate multiplier. For example, for k=8, the error rate of AM1 can be as high as 61.55%, but the MRED is only 1.87%, i.e., most of the errors are not significant.

B. Accuracy Evaluation of 16 × 16 Multipliers

Fig. 7 shows the Monte Carlo simulation results for the 16×16 designs of AM1, AM2, TAM1 and TAM2 with 10^8 random inputs. Likewise, the error decreases with an increasing number of bits used for error reduction. It is still true that AM2/TAM2 has a better accuracy than AM1/TAM1. Another observation is that AM1/AM2 has a better accuracy than TAM1/TAM2, as expected.

AM1/AM2 has a smaller NMED than TAM1/TAM2, however the difference is very small. This is because truncation of several LSBs does not significantly affect the overall NMED. For the same reason, the ME of TAM1/TAM2 is slightly higher than AM1/AM2. Yet for MRED, we can see that the difference between AM1/AM2 and TAM1/TAM2 becomes more significant because the relative error is easily affected by truncation. All these four approximate designs have high ERs (98%-100%), and TAM1/TAM2 results in nearly an ER of 100%. This is not surprising since 16×16 designs generate more error bits than 8×8 designs, and the truncation even generates more errors. However, the NMED and MRED are still kept very small.

V. Delay, Area and Power Evaluation

A. Analysis and Estimation

1) Delay Estimate: Based on the linear model of [26], the delays of a full adder (Fig. 8(a)) and the approximate adder cell (Fig. 8(b)) are approximately $4\tau_g$ and $3\tau_g$, respectively, where τ_g is an approximate "gate delay". The delay of an XOR (or XNOR) gate is $2\tau_g$ due to its higher complexity compared to an NAND (or NOR gate) [27].

For an $n \times n$ approximate multiplier (n is the power of 2), there are $m = \log_2 n$ stages in the partial product accumulation

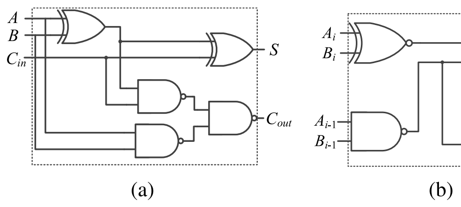

Fig. 8. (a) An exact full adder and (b) the approximate adder cell.

TABLE II
ESTIMATED DELAY OF THE PARTIAL PRODUCT ACCUMULATION TREE OF THE PROPOSED AND CONVENTIONAL MULTIPLIERS

n	8	16	32	64	2^l
$D_{AM1}(\tau_g)$	11	14	18	21	$3l + \log_2 l$
$D_{AM2}(\tau_g)$	15	18	24	27	$3l + 3\log_2 l$
$D_W(\tau_g)$	16	24	32	40	$\approx 6.5l$

tree. The first stage with 2^m rows of partial products are compressed to 2^{m-1} rows of partial products in the second stage and 2^{m-1} error vectors. These error vectors are then compressed (i.e., accumulated) using OR gates or approximate adders in a similar tree structure. Since the numbers of rows in the second partial product accumulation stage and the errors generated by the first stage are the same, it takes $m-1$ stages for both stages to be compressed to 1. Again, the number of error vectors generated by the second partial product accumulation stage is the same as the partial product rows in the third partial product accumulation stage; both of them require $m-2$ stages to compress the rows to 1. Thus, when an n-row partial product tree is compressed to 1 row, errors from the $\log_2 n$ stages are also compressed to $\log_2 n$ error vectors, provided that the delays for compressing two partial products and accumulating two error vectors are the same. As the delay of an OR gate is shorter than that of the approximate adder, fewer error vectors remain after $\log_2 n$ stages in AM1. For ease of analysis, the numbers of the remaining error vectors after $\log_2 n$ stages in both AM1 and AM2 are considered to be approximately $\log_2 n$. Then it takes $\lceil \log_2 \log_2 n \rceil$ stages to finally compress these $\log_2 n$ error vectors. Therefore, the delay of the proposed partial product accumulation scheme is modeled to be the sum of the delay of compressing the partial product tree and the delay to accumulate the remaining $\log_2 n$ error vectors, i.e.

$$D_{AMi} = (\log_2 n) \times 3\tau_g + \lceil \log_2 \log_2 n \rceil \times \tau_i, \qquad (13)$$

where $\tau_i = \tau_g$ (the delay of an OR gate for AM1) for $i = 1$ and $\tau_i = 3\tau_g$ (the delay of an approximate adder for AM2) for $i = 2$.

There are 4 compression stages in an 8×8 Wallace multiplier, and $\lfloor \log_{1.5} n \rfloor$ stages in an $n \times n$ Wallace multiplier ($n \geq 16$). Thus the delay of a Wallace tree is approximately given by [28]

$$D_W = 4 \lfloor \log_{1.5} n \rfloor \tau_g. \qquad (14)$$

Table II shows the delay of the partial product accumulation tree in both the proposed and Wallace multipliers. For a 16×16 multiplier, the delay of an exact multiplier tree is nearly $1.5\times$ as large as the delay of the proposed multiplier tree. As the size of the multiplier increases, this factor is approximately 2. In the Wallace multiplier that is optimized for speed [27], the partial product accumulation delay is improved for up to 30% by optimizing the signal connections between full adders. As a result, the proposed partial product accumulation design is 29% faster than the optimized Wallace multiplier. In summary, the proposed multiplier can significantly reduce the delay of the partial product accumulation tree, which scales with the size of the multiplier.

In an $n \times n$ Wallace multiplier, a final $2n$-bit CPA is required for adding the resultant two partial product rows. The entire delay of a Wallace multiplier is given by the addition of the delays caused by the Wallace tree and the final CPA. In the proposed design, however, the partial products are compressed to one row and thus, only a $(k-1)$-bit CPA ($k < 2n$) is required to compensate the error. Thus, the proposed approximate multiplier is faster than a Wallace multiplier when the same adder design is used for final addition.

2) Area Estimate: Let the area of a basic gate be α_g, and the area for an XOR (or XNOR) gate be $2\alpha_g$ [29]. Then, the area of a full adder cell is $7\alpha_g$, and the area of the approximate adder cell is $5\alpha_g$. If the error signal E_i is not required, the circuit area for generating a sum S_i is $4\alpha_g$, i.e., an NOR gate is not needed.

As the number of partial product rows is reduced by 1 by using an $(n-1)$-bit approximate adder, $(n-1)$ $(n-1)$-bit approximate adders are required to compress the n partial product rows to one row. Also, $(n-1)$ error vectors are generated, because each approximate adder produces an error vector. The number of OR gates (or approximate adders) used for error accumulation is determined by the number of MSBs used for error reduction (i.e., k). Thus, the area of the proposed partial product accumulation scheme is estimated to be

$$A_{AMi} = (n-1)^2 \times 4\alpha_g + \alpha_i, \qquad (15)$$

where α_i is the area of the error generation and accumulation circuit in AMi ($i = 1$ or 2).

In an $n \times n$ Wallace multiplier, a full adder compresses three partial products to two, i.e., one bit is reduced by using a full adder. Thus, $(n-2)$ rows of full adders are used to compress the n partial product rows to two; each row consists of approximately $(n-1)$ full adders. The area of the Wallace tree is given by

$$A_W = 7(n-2)(n-1)\alpha_g. \qquad (16)$$

Consider $n = 8$ as an example, Table III shows the estimated areas of the Wallace tree and the partial product accumulation

TABLE III
ESTIMATED AREA OF PARTIAL PRODUCT ACCUMULATION TREE FOR THE PROPOSED AND CONVENTIONAL 8 × 8 MULTIPLIERS

k	4	6	8	14
$A_{AM1}(\alpha_g)$	205	221	245	281
$A_{AM2}(\alpha_g)$	213	264	305	385
$A_W(\alpha_g)$	294	294	294	294

tree of the proposed multipliers using different numbers of MSBs for error reduction. According to the estimate, the partial product accumulation tree of AM1 has a smaller area than an Wallace tree, whereas the area of AM2's partial product accumulation tree is larger than an Wallace tree when the number of MSBs used for error reduction is larger than 8. Note that the final CPA used for error reduction in the proposed multiplier has a smaller area than a Wallace multiplier. Thus, to achieve a similar area as a Wallace multiplier, the number of MSBs used for error reduction in AM2 can be larger than 8.

3) Power Estimate: The power consumption of a CMOS circuit consists of short-circuit power, leakage power and dynamic power [26]. Compared to the dynamic power, the short-circuit and leakage powers are relatively small and vary with device fabrication. Dynamic power is dissipated for charging or discharging the load capacitance when the output of a CMOS circuit switches. By using a probabilistic power analysis, the average dynamic power of a circuit is given by [30]

$$P_{avg} = f_{clk} \cdot V_{dd}^2 \sum_{i=1}^{N} C_L(x_i) \cdot \alpha_{0\to 1}(x_i), \tag{17}$$

where f_{clk} is the operating clock frequency of the circuit, V_{dd} is the supply voltage, N is the number of nodes in the circuit, $C_L(x_i)$ is the load capacitance at node x_i, and $\alpha_{0\to 1}(x_i)$ is the probability of the logic transition from 0 to 1 at node x_i. $\alpha_{0\to 1}(x_i)$ is computed by

$$\alpha_{0\to 1}(x_i) = P_s(x_i)P_s(\bar{x_i}), \tag{18}$$

where $P_s(x_i)$ is the signal probability at node x_i; it is defined as the probability of a high signal value occurring at x_i.

As the basic components of the Wallace and the proposed multipliers, the full adder and the proposed approximate adder are analyzed using (17). In (17), f_{clk} and V_{dd} are the same for the two components, $C_L(x_i)$ depends on the fabrication. Thus, the difference in dynamic power dissipation between these two components is mainly caused by $\alpha_{0\to 1}(x_i)$.

Assume that 0 and 1 are equally likely to occur in each input bit of the multiplication, i.e., the signal probability of an input bit is 0.5, the partial product generated by a 2-input AND gate has a signal probability of $0.5 \times 0.5 = 0.25$. For ease of calculation, the input partial products to the full adder and the proposed approximate adder are assumed to be mutually independent. For the full adder in Fig. 8(a), the signal probabilities of the two outputs are computed as per their truth tables, i.e., $P_s(S) = 7/16$ and $P_s(C_{out}) = 5/32$. Thus, $\alpha_{0\to 1}(S) = 7/16\times(1-7/16) = 0.246$ and $\alpha_{0\to 1}(C_{out}) = 0.132$. Compared to the full adder, the proposed approximate adder in Fig. 8(b) has a similar signal probability at the sum output, i.e., $P_s(S_i) = 53/128$, while $P_s(E_i) = 3/128$ that is significantly lower than $P_s(C_{out})$. So, $\alpha_{0\to 1}(S_i) = 0.243$ and $\alpha_{0\to 1}(E_i) = 0.023$.

As $P_s(S_i) < P_s(S)$ and $P_s(E_i) < P_s(C_{out})$, the dynamic power dissipated at the two outputs of the proposed approximate adder is lower than a full adder. As for the internal nodes, the full adder has one more node than the proposed approximate adder. Thus, the proposed approximate adder consumes lower dynamic power than a full adder. Moreover, the dynamic power consumed by the error vector accumulation circuit is very low due to the low switching activity at E_i. Consequently, the proposed approximate multiplier is more power-efficient than a Wallace multiplier.

B. Simulation Results

1) 8×8 Multipliers: AM1 has shown advantages in speed and power consumption compared to a Wallace multiplier for FPGA implementations, as discussed in [23]. A more detailed discussion of the circuit implementations is pursued next. Designs for 8×8 AM1 using $4, 5, \ldots, 9$ MSBs for error reduction, 8×8 AM2 using $4, 5, \ldots, 9$ MSBs for error reduction, and the 8×8 optimized Wallace multiplier [27] have been implemented in VHDL and synthesized by using the Synopsys Design Compiler (DC) with an industrial 28 nm CMOS process. Simulations are performed at a temperature of 25°C and a supply voltage of 1V. The modules for implementing the multiplier circuits are taken from the 28 nm library as C32_SC_12_CORE_LR_tt28_1.00V_25C. The critical path delays of these multipliers are reported by the Synopsys DC tool. The power dissipation is found by the PrimeTime-PX tool using 10 million random input combinations with a clock period of 2 ns. The delay, area, power and power-delay product (PDP) are shown in Fig. 9, where the area is optimized to the smallest value for the results in (a), (b), (c) and (d), and the critical path delay is constrained to the smallest value without timing violation for the results in (e), (f), (g) and (h). The reported power consumption is the total power, i.e., the sum of the dynamic and static powers.

Figs. 9(a) and (e) indicate that the proposed approximate multiplier designs have shorter delay than the accurate Wallace multiplier. The critical path delay of AM1 and AM2 increases with the number of MSBs employed in the error reduction process. At the same number of MSBs in error reduction, AM1 shows a shorter delay than AM2; this occurs because AM1 uses a simpler OR-gate based error reduction scheme. Specifically, the delays for 8/4 AM1, 8/4 AM2 and the Wallace multiplier are 0.40 (0.16) ns, 0.43 (0.16) ns and 1.08 (0.40) ns, respectively, for the area (delay)-optimized circuits. Thus AM1 and AM2 with 4-bit error reduction are faster by 63% and 60% than the Wallace multiplier when optimized for area, while they are faster by 60% when optimized for delay. For the 8-bit error reduction scheme, these values are 22% (28%) and 19% (5%), respectively, for the area (delay)-optimized circuits.

The power dissipation and area of the multipliers show the same trend as the delay (Figs. 9(b), (f) and (c), (g)). For the area-optimized circuits, 8/4 AM1 and 8/4 AM2 save as much as 42% in power and 34% in area compared with the Wallace multiplier. The power improvements of AM1 and AM2 are 21% and 17% when 8 MSBs are used for error reduction. For the delay-optimized circuits, 8/4 AM1 and 8/4 AM2 consume a lower power by 53% and a smaller area by 38% than the Wallace multiplier. For the 8-bit error reduction scheme, the power savings of AM1 and AM2 are approximately 20%. The area-optimized 8/4 AM1 and AM2 use a smaller area by nearly 23% (by 38% for delay-optimized circuits) than the accurate design. However, the area of AM2 is larger than the

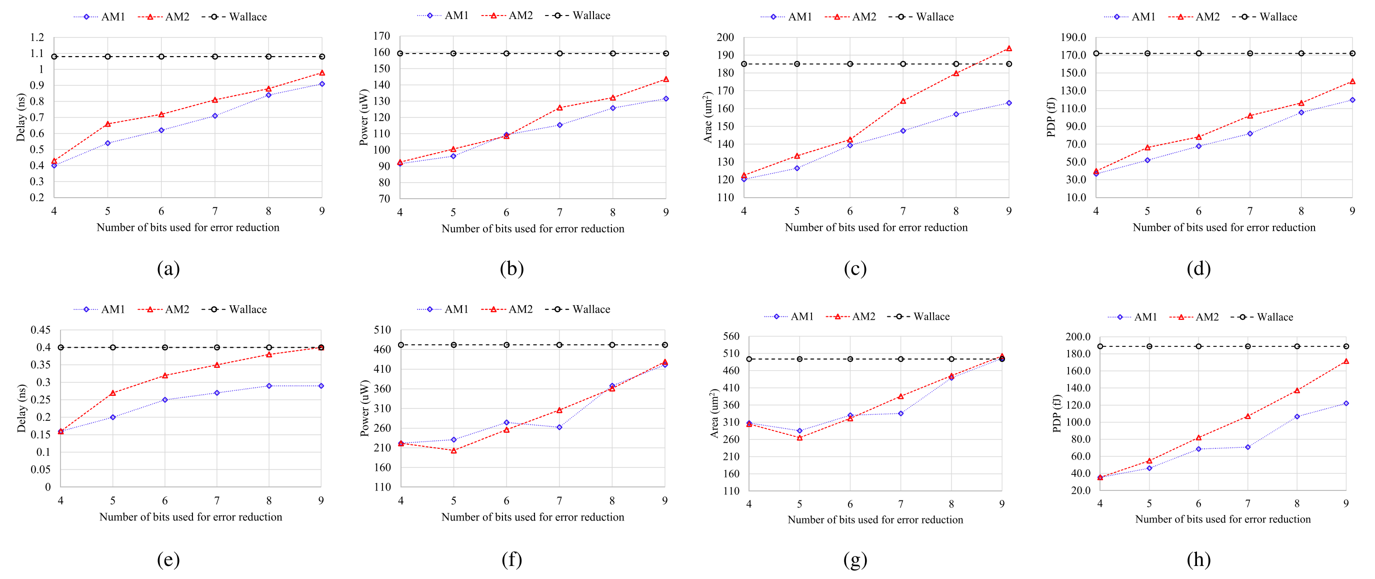

Fig. 9. Delay, power and area comparisons of proposed 8 × 8 approximate and Wallace multipliers. "Wallace" indicates the accurate 8 × 8 Wallace multiplier, and the X-axis is not applicable for it. (a) Delay (optimized for area). (b) Power (optimized for area). (c) Area (optimized for area). (d) PDP (optimized for area). (e) Delay (optimized for delay). (f) Power (optimized for delay). (g) Area (optimized for delay). (h) PDP (optimized for delay).

Wallace multiplier when the number of error reduction bits is larger than 8. Figs. 9(d) and (h) show that the PDPs of AM1 and AM2 are smaller than the Wallace multiplier by 38% to 81% and 27% to 81%, respectively, with 4 to 8-bit error reduction.

2) 16 × 16 Multipliers: Similarly, designs for 16 × 16 AM1, AM2, TAM1 and TAM2 are implemented in VHDL and synthesized by using the Synopsys DC tool with the same technique and configurations as the 8 × 8 designs. Different from the 8 × 8 designs, the power for the 16 × 16 designs is evaluated under a clock period of 4 *ns*. Also, the optimized 16 × 16 Wallace multiplier [27] is synthesized. The reported results of the critical path delay, power consumption and area utilization are shown in Fig. 10, where the number of bits used for error reduction for the proposed designs is from 10 to 16, and these numbers are not applicable for the accurate Wallace multiplier.

Fig. 10 shows that the delays of AM1, AM2, TAM1 and TAM2 are shorter than the Wallace multiplier by approximately 24% to 50% when optimized for area. However, AM2 and TAM2 are slower than the Wallace multiplier when the designs are synthesized for the minimal delay, while TAM1 is faster by more than 25%. The power dissipations of 16 × 16 AM1 and AM2 are very close for the same number of bits used for error reduction (Figs. 10(b) and (f)). They save from 18% to 35% in power compared with the Wallace multiplier when optimized for area, while this value is from 20% to 60% for the delay-optimized circuits. Similarly, TAM1 and TAM2 consume a lower power by 50% to 66% (for optimized area) and by 40% to 66% (for optimized delay). The results for area show a similar trend. Compared to the Wallace multiplier, TAM1 and TAM2 save from 38% to 62% area when area is optimized, while the area is reduced by 32% to 60% when delay is optimized. For the area-optimized circuits, the area reduction is between 5% and 30% for AM1 and AM2; it decreases with the increase of the number of MSBs used for error reduction. The results in Figs. 10(d) and (h) show that TAM1 incurs a smaller PDP by 61% to 83% than the Wallace multiplier, and this value is between 32% and 79% for TAM2.

VI. Comparison With Existing Approximate Multipliers

Next, 8 × 8 AM1 and AM2 are compared with three other approximate multipliers of the same size: the ETM [16], the UDM [14] and the SSM [17], as illustrated in Fig. 11. The accuracy characteristics are obtained by Monte Carlo simulation with 10^8 random input combinations. The circuit characteristics are obtained by synthesizing all approximate designs using the same tool, process, temperature and supply voltage with the same input combinations and clock period as detailed in the previous section. Moreover, the PDP and area-delay product (ADP) are calculated to better assess performance at the circuit level. In this comparison, ETM and SSM with 4, 5 and 6 MSBs as the accurate multiplication part are considered and they are referred to as ETM*k* and SSM*k* ($k < 8$ is the width of the accurate part). The results are shown in Fig. 11 for each of the metrics. There is only one configuration for UDM, so the values for it are constant for each metric.

Among these five multipliers, AM1 has the lowest PDP and ADP when a similar MRED, NMED or ER is considered. AM2 also performs better than the other approximate multipliers. ETM has the lowest accuracy in terms of MRED and NMED, because ETM uses a simple partition scheme and as reported in [16], it saves significant power. Likewise, SSM shows very high values of MRED, NMED and ER. As ETM and SSM utilize an accurate multiplier with size larger than

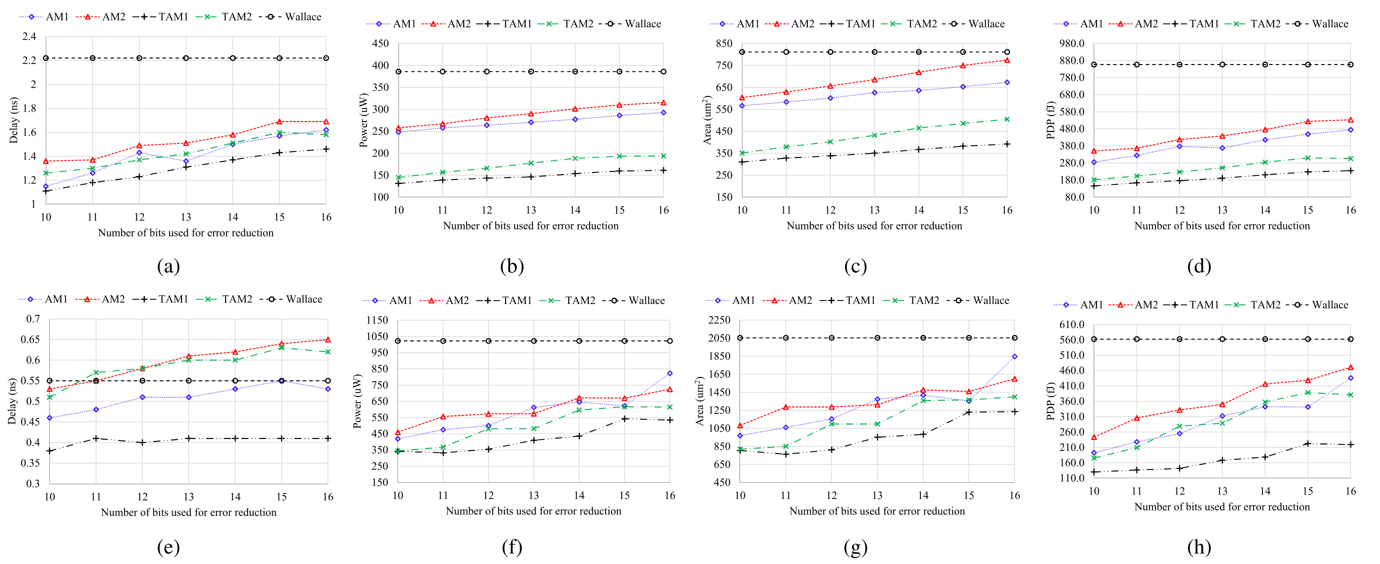

Fig. 10. Delay, power and area comparisons of the proposed 16 × 16 approximate and the optimized Wallace multipliers. "Wallace" indicates the accurate 16 × 16 Wallace multiplier, and the X-axis is not applicable for it. (a) Delay (optimized for area). (b) Power (optimized for area). (c) Area (optimized for area). (d) PDP (optimized for area). (e) Delay (optimized for delay). (f) Power (optimized for delay). (g) Area (optimized for delay). (h) PDP (optimized for delay).

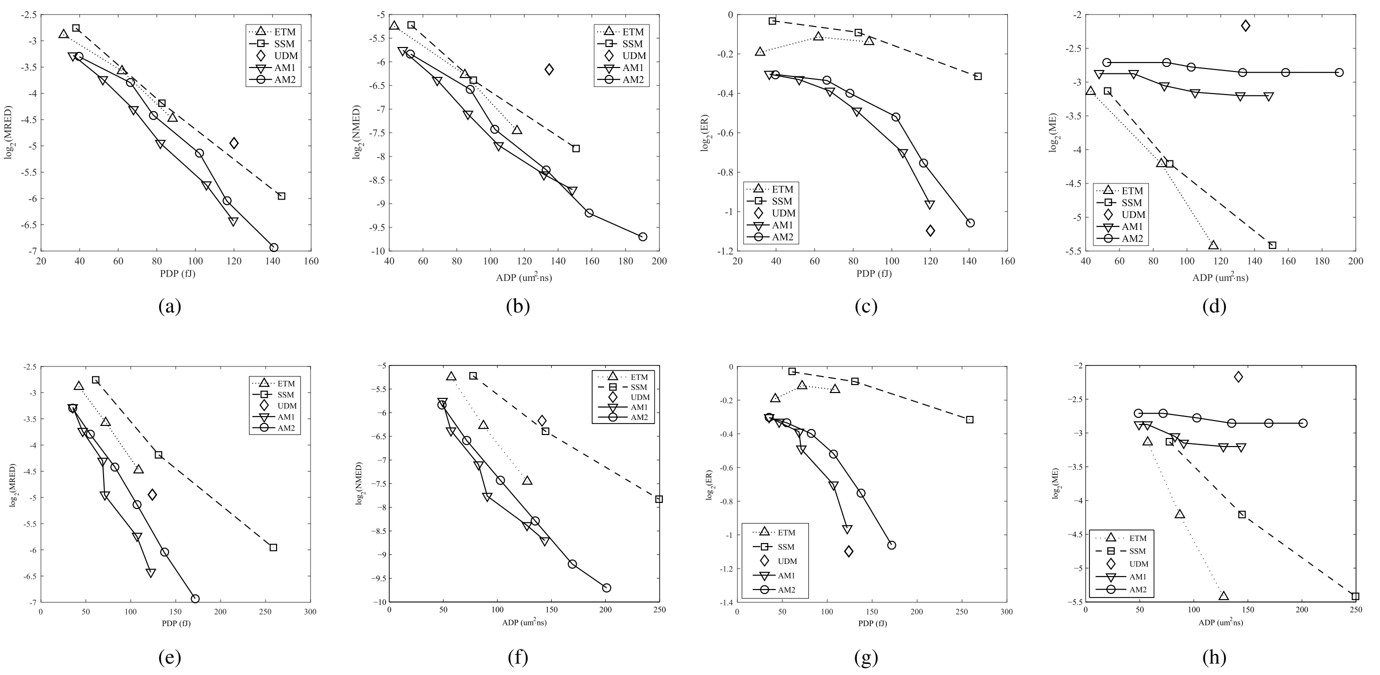

Fig. 11. Comparison of accuracy and hardware-efficiency among five approximate 8 × 8 multipliers. The number of MSBs used for error reduction for AM1 and AM2 ranges from 4 to 9 from left to right. The width of the accurate multiplier for ETM and SSM ranges from 4 to 6 from left to right. (a) PDP (area-optimized) vs. MRED. (b) ADP (area-optimized) vs. NMED. (c) PDP (area-optimized) vs. ER. (d) ADP (area-optimized) vs. ME. (e) PDP (delay-optimized) vs. MRED. (f) ADP (delay-optimized) vs. NMED. (g) PDP (delay-optimized) vs. ER. (h) ADP (delay-optimized) vs. ME.

half of the original design, they attain the smallest values of ME (Figs. 11(d) and (h)). The ME for AM2 is higher than AM1, ETM and SSM because of the approximate adders used in the error accumulation tree (Fig. 4). Specifically, the approximate adders in stages 2 and 3 generate not only sums but also error vectors. As only the sums are used for the final error compensation, the omitted error vectors at the higher bit positions can lead to very large errors. Although the ME values for AM1 and AM2 are not as low as those of ETM and SSM, the small values of NMED and MRED indicate that the probability of occurrence of a large ED is very low. UDM has the lowest ER but the largest ME with a moderate PDP and ADP.

Fig. 12 shows the comparison results of 16×16 approximate multipliers for accuracy and hardware-efficiency. In addition to ETM, SSM and UDM, another high-performance, area and

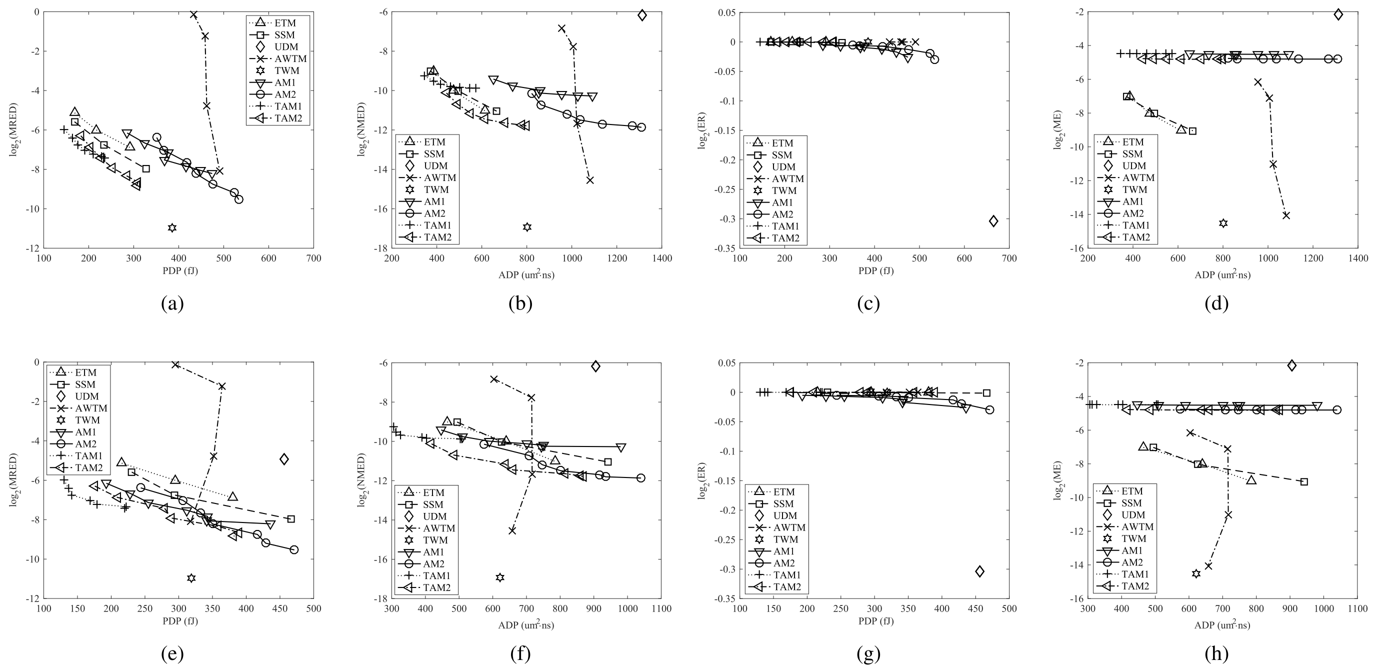

Fig. 12. Comparison of accuracy and hardware-efficiency of approximate 16 × 16 multipliers. The width of the accurate multiplier for ETM and SSM is from 8 to 10 from left to right. The parameter for AWTM is the mode number (1 to 4) from left to right. (a) PDP (area-optimized) vs. MRED. (b) ADP (area-optimized) vs. NMED. (c) PDP (area-optimized) vs. ER. (d) ADP (area-optimized) vs. ME. (e) PDP (delay-optimized) vs. MRED. (f) ADP (delay-optimized) vs. NMED. (g) PDP (delay-optimized) vs. ER. (h) ADP (delay-optimized) vs. ME.

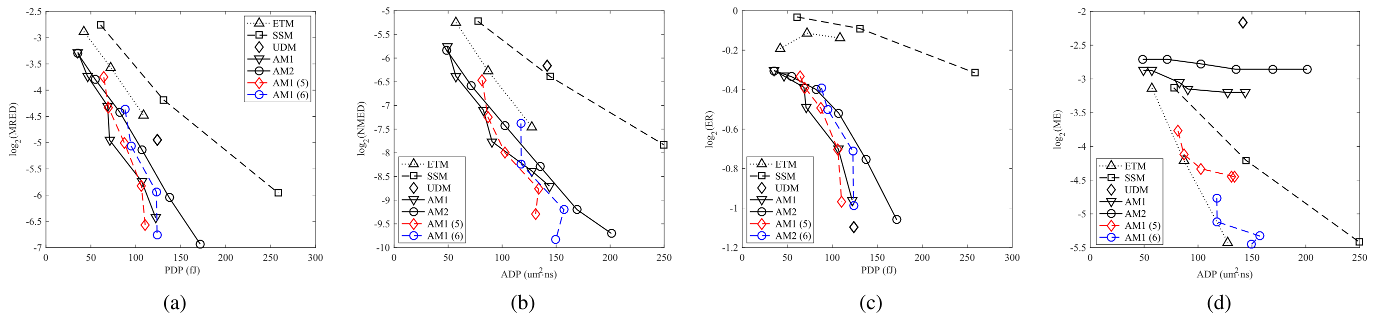

Fig. 13. Comparison of accuracy and hardware-efficiency (delay-optimized) of improved 8 × 8 AM1 with other designs. The number of MSBs used for error reduction for AM1 and AM2 ranges from 4 to 9, and the width of the accurate multiplier for ETM and SSM is from 4 to 6, from left to right. AM1 (5) and AM1 (6) are AM1's with 5 and 6 MSBs of errors that are correctly accumulated. Thus, the number of MSBs used for error reduction for AM1 (5) is from 5 to 9, and it is from 6 to 9 for AM1 (6). (a) PDP vs. MRED. (b) ADP vs. NMED. (c) PDP vs. ER. (d) ADP vs. ME.

power efficient 16 × 16 approximate multiplier, AWTM [15], is considered in this comparison. Also, the truncated Wallace multiplier (referred to as TWM) that truncates half partial products with data-dependent error compensation is compared [31]. Fig. 12(c) shows that all the multipliers have close to 100% ERs except for UDM that has a relatively lower ER. Among the 16×16 approximate multipliers, TAM1 and TAM2 perform very well in terms of MRED and NMED for a similar PDP or ADP, while AM1, AM2 and UDM are useful when most of the input operands are very small. AWTM mode 4 is also a good design with small values of MRED and NMED, as well as moderate PDP and ADP. TWM with low MRED, NMED and ME has a very high accuracy, whereas its PDP and ADP are relatively higher compared to TAM1. Fig. 12(d) shows that TAM1 (TAM2) has a similar ME with AM1 (AM2), which indicates that truncation does not significantly affect the ME.

As per the comparison, the large MEs are the main drawbacks of the proposed designs, as shown in Figs. 11(d) and (h) and Figs. 12(d) and (h). This is because some errors at the higher bit positions are not correctly accumulated by using OR gates and the proposed approximate adders. Therefore, to decrease the MEs of the proposed design, the errors at the higher bit positions should be accumulated using accurate full or half adders. The efficiency of this methodology is evaluated by simulating the 8 × 8 AM1 with 5 and 6 MSBs of errors that are correctly accumulated (the other MSBs are accumulated by using OR gates when the number of MSBs used for error reduction is larger than 5 and 6, respectively); they are referred to as AM1 (5) and AM1 (6). The comparison results are shown in Fig. 13. Fig. 13(d) shows that the ME of AM1 is significantly decreased by increasing the number of accurately accumulated MSBs, with slightly increased ADP and PDP. However, the MRED, NMED and ER of AM1 are

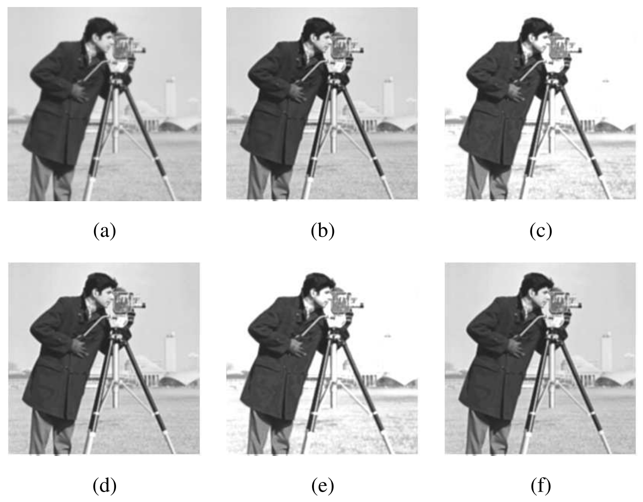

Fig. 14. Images sharpened using the proposed multipliers. (a) Blurred input. (b) Accurate multiplier. (c) 8/4 AM1. (d) 8/8 AM1. (e) 8/4 AM2. (f) 8/8 AM2.

TABLE IV
PSNR OF IMAGE PROCESSING APPLICATIONS FOR AM1 AND AM2 (dB)

Image Processing	Image Sharpening			Image Smoothing		
Configuration	8/4	8/6	8/8	8/4	8/6	8/8
AM1	18.45	25.80	39.89	30.64	33.44	46.70
AM2	18.45	25.80	40.18	30.64	33.44	46.70

only slightly lowered, as shown in Figs. 13(a) (b) and (c). Thus, some MSBs should be accumulated using accurate adders when the ME is critical for an application; otherwise, OR gates or approximate adders with lower hardware overhead are preferred.

VII. Image Processing Applications

A. Image Processing With Proposed Multipliers

Approximate circuits can be used in error-tolerant applications such as image processing; image sharpening and smoothing applications are studied next. Since multiplication is the arithmetic operation under investigation, accurate multipliers are replaced by the proposed approximate multipliers (i.e., AM1 and AM2). All other processing steps (such as addition) are kept accurate.

The sharpening algorithm of [32] is simulated using both exact and approximate multipliers (i.e., AM1 and AM2). In the results shown in Fig. 14, approximate multipliers with different numbers of MSBs for error reduction are evaluated and an improvement in performance is achieved when the number of MSBs is increased for further error reduction. The degradation in image quality is evident when 4 MSBs are used for error reduction for both AM1 and AM2. However, for an 8-bit error reduction in AM1 and AM2, there is no visually distinguishable difference with the exact sharpening result.

The image smoothing algorithm is given by [33]:

$$Y(x,y) = \frac{1}{60}\sum_{m=-2}^{2}\sum_{n=-2}^{2} X(x-m, y-n)Mask(m,n), \quad (19)$$

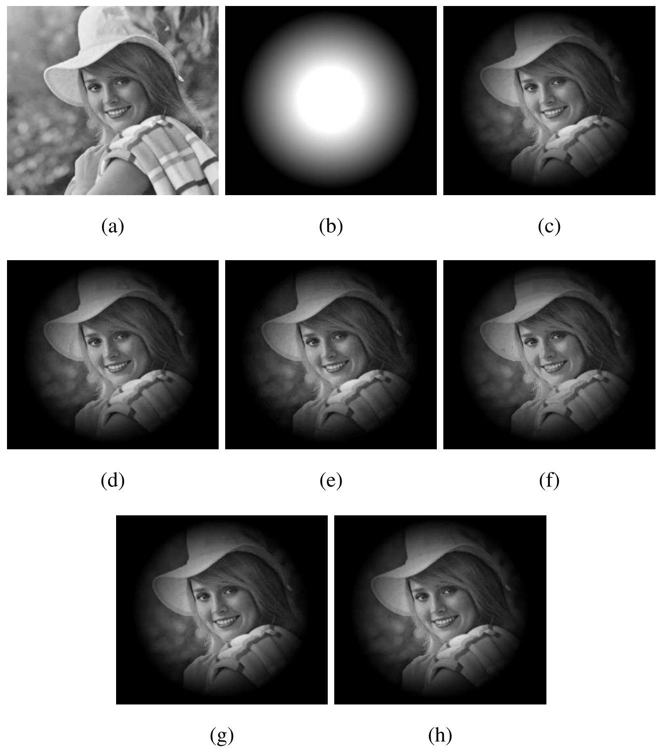

Fig. 15. Images multiplied by different multipliers. (a) Original image 1. (b) Original image 2. (c) Accurate multiplier. (d) 8/6 AM1. (e) 8/5 AM2. (f) UDM. (g) ETM5. (h) SSM5.

TABLE V
PSNR OF IMAGE MULTIPLICATION OF FIVE DIFFERENT APPROXIMATE MULTIPLIERS (dB)

Multiplier	8/6 AM1	8/5 AM2	UDM	ETM5	SSM5
PSNR	39.21	37.60	34.56	37.53	37.40

where X is the input image, Y is the output smoothed image, and $Mask$ is a 5×5 matrix given by:

$$Mask = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 4 & 4 & 4 & 1 \\ 1 & 4 & 12 & 4 & 1 \\ 1 & 4 & 4 & 4 & 1 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}.$$

The peak signal-to-noise ratio (PSNR) is used for comparison of the difference between the images obtained by the accurate and approximate multiplications. Table IV shows the PSNR values with respect to different numbers of MSBs for error reduction in the proposed approximate multiplier. For example, the resulting image by an 8/8 AM1 has a PSNR of 39.89 dB for image sharpening and 46.70 dB for image smoothing; this is generally considered to be a good match with the accurately processed image. Since the result of an approximate multiplication is processed by an accurate division for both image sharpening and smoothing applications, the error in the approximate multiplication is attenuated. Therefore, the differences in the PSNRs for AM1 and AM2 are very small and, thus, difficult to be observed by a 2-digit precision. However, there is a 0.3 dB difference between the

PSNRs for AM1 and AM2 with 8-bit error reductions for the image sharpening application.

B. Comparison With Existing Approximate Multipliers

To evaluate the performance of each approximate multiplier, image multiplication is selected because it directly employs multiplication without any other operations. As AM1, AM2, ETM and SSM have different configurations, configurations with similar PDP values are selected for image multiplication, i.e., 8/6 AM1, 8/5 AM2, SSM5 and ETM5, are considered (Fig. 11). The resulting images by UDM (Fig. 15) show a reduction in quality, while there are few visible flaws for the image processed by the other approximate multipliers. In terms of PSNR, 8/6 AM1 achieves the highest value (Table V), while UDM has the lowest. The values of PSNR for ETM5 and SSM5 are the second lowest. These results are consistent with the NMED trend of the approximate multipliers. It also indicates that an approximate multiplier with a high ME does not necessarily result in a poor image quality in image multiplication as long as its NMED is low.

VIII. Conclusion

This paper proposes a high-performance and low-power approximate partial product accumulation tree for a multiplier using a newly designed approximate adder. The proposed approximate adder ignores the carry propagation by generating both an approximate sum and an error signal. OR gate and approximate adder based error reduction schemes are utilized, yielding two different approximate 8×8 multiplier designs: AM1 and AM2. Moreover, modifications are made on the error reduction schemes for 16×16 multiplier designs, such that TAM1 and TAM2 are obtained by truncating 16 LSBs of the partial products. The proposed approximate multipliers have been shown to have a lower power dissipation than an exact Wallace multiplier optimized for speed. Functional analysis has shown that on a statistical basis, the proposed multipliers have very small error distances and thus, they achieve a high accuracy. Simulation has also shown that AM2 has a higher accuracy than AM1 at the cost of a longer delay and a higher power consumption. Truncation-based designs (TAM1 and TAM2) achieve a significant improvement in power and area with a small degradation in NMED. The proposed approximate multipliers improve over previous approximate designs especially in accuracy. While previous designs focus on reducing both delay and power with often unsatisfying accuracy, the proposed designs achieve excellent delay and power reductions with a high accuracy. The application of the proposed multipliers to image sharpening and smoothing has shown that the proposed designs are very competitive in performance with their accurate counterpart.

References

[1] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proc. 18th IEEE Eur. Test Symp.*, May 2013, pp. 1–6.

[2] S.-L. Lu, "Speeding up processing with approximation circuits," *Computer*, vol. 37, no. 3, pp. 67–73, Mar. 2004.

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

[4] N. Zhu, W. L. Goh, and K. S. Yeo, "An enhanced low-power high-speed adder for error-tolerant application," in *Proc. 12th Int. Symp. Integr. Circuits*, Dec. 2009, pp. 69–72.

[5] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 4, pp. 850–862, Apr. 2010.

[6] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "IMPACT: IMPrecise adders for low-power approximate computing," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design*, Aug. 2011, pp. 409–414.

[7] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Proc. Design Automat. Conf.*, Jun. 2012, pp. 820–825.

[8] K. Du, P. Varman, and K. Mohanram, "High performance reliable variable latency carry select addition," in *Proc. Design, Automat. Test Eur. Conf. Exhib.*, Mar. 2012, pp. 1257–1262.

[9] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *IEEE Trans. Comput.*, vol. 62, no. 9, pp. 1760–1771, Jun. 2012.

[10] J. Huang, J. Lach, and G. Robins, "A methodology for energy-quality tradeoff using imprecise hardware," in *Proc. Design Automat. Conf.*, Jun. 2012, pp. 504–509.

[11] J. Miao, K. He, A. Gerstlauer, and M. Orshansky, "Modeling and synthesis of quality-energy optimal approximate adders," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2012, pp. 728–735.

[12] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "MACACO: Modeling and analysis of circuits for approximate computing," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2011, pp. 667–673.

[13] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han, "A review, classification, and comparative evaluation of approximate arithmetic circuits," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 4, 2017, Art. no. 60.

[14] P. Kulkarni, P. Gupta, and M. D. Ercegovac, "Trading accuracy for power in a multiplier architecture," *J. Low Power Electron.*, vol. 7, no. 4, pp. 490–501, 2011.

[15] K. Bhardwaj, P. S. Mane, and J. Henkel, "Power- and area-efficient approximate wallace tree multiplier for error-resilient systems," in *Proc. 15th Int. Symp. Qual. Electron. Design*, Mar. 2014, pp. 263–269.

[16] K. Y. Kyaw, W. L. Goh, and K. S. Yeo, "Low-power high-speed multiplier for error-tolerant application," in *Proc. IEEE Int. Conf. Electron Devices Solid-State Circuits*, Dec. 2010, pp. 1–4.

[17] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim, "Energy-efficient approximate multiplication for digital signal processing and classification applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 6, pp. 1180–1184, Jun. 2015.

[18] Y.-H. Chen and T.-Y. Chang, "A high-accuracy adaptive conditional-probability estimator for fixed-width booth multipliers," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 59, no. 3, pp. 594–603, Mar. 2012.

[19] B. Shao and P. Li, "Array-based approximate arithmetic computing: A general model and applications to multiplier and squarer design," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 4, pp. 1081–1090, Apr. 2015.

[20] H. Jiang, J. Han, F. Qiao, and F. Lombardi, "Approximate radix-8 booth multipliers for low-power and high-performance operation," *IEEE Trans. Comput.*, vol. 65, no. 8, pp. 2638–2644, Aug. 2016.

[21] K. Nepal, Y. Li, R. I. Bahar, and S. Reda, "ABACUS: A technique for automated behavioral synthesis of approximate computing circuits," in *Proc. Design, Automat. Test Eur. Conf. Exhib.*, Mar. 2014, pp. 1–6.

[22] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan, "ASLAN: Synthesis of approximate sequential circuits," in *Proc. Design, Automat. Test Eur. Conf. Exhib.*, Mar. 2014, pp. 1–6.

[23] C. Liu, J. Han, and F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *Proc. Design, Automat. Test Eur. Conf. Exhib.*, Mar. 2014, pp. 1–4.

[24] B. Parhami, *Computer Arithmetic*. London, U.K.: Oxford Univ. Press, 2000.

[25] M. A. Breuer, "Intelligible test techniques to support error-tolerance," in *Proc. 13th Asian Test Symp.*, Nov. 2004, pp. 386–393.

[26] N. Weste and H. David, *CMOS VLSI Design: A Circuits and Systems Perspective*, 3rd ed. London, U.K.: Pearson, 2005.

[27] V. G. Oklobdzija, D. Villeger, and S. S. Liu, "A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach," *IEEE Trans. Comput.*, vol. 45, no. 3, pp. 294–306, Mar. 1996.
[28] K. C. Bickerstaff, E. E. Swartzlander, and M. J. Schulte, "Analysis of column compression multipliers," in *Proc. 15th IEEE Symp. Comput. Arithmetic*, Jun. 2001, pp. 33–39.
[29] K. C. Bickerstaff, M. J. Schulte, and E. E. Swartzlander, Jr., "Parallel reduced area multipliers," *J. VLSI Signal Process. Syst. Signal, Image Video Technol.*, vol. 9, no. 3, pp. 181–191, 1995.
[30] Y.-K. Cheng, C.-H. Tsai, C.-C. Teng, and S.-M. Kang, *Electrothermal Analysis of VLSI Systems*. New York, NY, USA: Springer, 2002.
[31] E. J. King and E. E. Swartzlander, "Data-dependent truncation scheme for parallel multipliers," in *Proc. 31st Conf. Rec. Asilomar Conf. Signals, Syst. Comput.*, vol. 2, Nov. 1997, pp. 1178–1182.
[32] M. S. K. Lau, K.-V. Ling, and Y.-C. Chu, "Energy-aware probabilistic multiplier: Design and analysis," in *Proc. Int. Conf. Compil., Archit., Synthesis Embedded Syst.*, 2009, pp. 281–290.
[33] H. R. Myler and A. R. Weeks, *The Pocket Handbook of Image Processing Algorithms in C*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1993.

Honglan Jiang (S'14) received the B.Sc. and master's degrees in instrument science and technology from the Harbin Institute of Technology, Harbin, Heilongjiang, China, in 2011 and 2013, respectively. She is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada. Her current research interests are approximate computing and stochastic computing.

Cong Liu received the B.Sc. degree in automation from Tsinghua University, Beijing, China, in 2012, and the master's degree in integrated circuits and systems from the University of Alberta, Edmonton, AB, Canada, in 2014. He is currently a Software Development Engineer with Amazon.

Fabrizio Lombardi (M'81–SM'02–F'09) received the B.Sc. degree (Hons.) in electronic engineering from the University of Essex, U.K., in 1977, the master's degree in microwaves and modern optics and the Diploma degree in microwave engineering from the Microwave Research Unit, University College London, in 1978, and the Ph.D. degree from the University of London in 1982. He is currently the International Test Conference (ITC) Endowed Chair Professorship with Northeastern University, Boston, USA. His research interests are bio-inspired and nano manufacturing/computing, VLSI design, testing, and fault/defect tolerance of digital systems. He has extensively published in these areas and coauthored/edited seven books. He was the Editor-In-Chief of the IEEE TRANSACTIONS ON COMPUTERS from 2007 to 2010 and the inaugural Editor-in-Chief of the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING from 2013 to 2017. He is the Editor-in-Chief of the IEEE TRANSACTIONS ON NANOTECHNOLOGY.

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 an Associate Professor with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada. His research interests include approximate computing, stochastic computation, reliability and fault tolerance, nanoelectronic circuits and systems, and novel computational models for nanoscale and biological applications. He and his co-authors received the Best Paper Award at the International Symposium on Nanoscale Architectures (NanoArch 2015) and the Best Paper Nominations at the 25th Great Lakes Symposium on VLSI (GLSVLSI 2015), the 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 in 2005. He is currently an Associate Editor of the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING, the IEEE TRANSACTIONS ON NANOTECHNOLOGY, and *Microelectronics Reliability*. He served as the General Chair for GLSVLSI 2017 and the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT 2013), and the Technical Program Committee Chair for GLSVLSI 2016 and DFT 2012.