Probabilistic Error Analysis of Approximate Adders and Multipliers

Sana Mazahir, Muhammad Kamran Ayub, Osman Hasan, and Muhammad Shafique

Abstract Approximate adders and multipliers are widely being advocated to be used in error resilient applications. A very important performance metric in this regard is the probability of occurrence of error in these arithmetic circuits as this allows us to choose the most efficient configuration of an adder or multiplier for a given application. In this chapter, we present an analytical error analysis approach for approximate adders, which comprise of subadder units, and recursive approximate multipliers with approximate partial products. We also derive Probability Mass Function (PMF) of error for both of the considered adder and multiplier models. The results show that the proposed analysis serves as an effective tool for predicting, evaluating and comparing the accuracy of various approximate adders and multipliers. For illustration purposes, we also show that the comparative performance of different approximate adders and multipliers can be correctly predicted in practical applications of image processing.

1 Introduction

Approximate computing [1] has introduced new avenues of hardware optimizations in digital designs for computationally extensive applications, which can afford compromising the exactness in the traditional arithmetic as a trade-off of improved power, speed or area. There are many such resource and power-hungry applications,

S. Mazahir
Georgia Institute of Technology, Atlanta, USA, e-mail: smazahir3@gatech.edu

M. K. Ayub and O. Hasan
School of Electrical Engineering and Computer Science, National University of Sciences and Technology (NUST), Islamabad, Pakistan e-mail: {14mseemayub,osman.hasan}@seecs.edu.pk

M. Shafique
Institute of Computer Engineering, Vienna University of Technology (TU Wien), Vienna, Austria e-mail: muhammad.shafique@tuwien.ac.at

e.g. big data analytics, image/video processing, data mining, computer vision, artificial intelligence, deep learning networks etc., which are inherently error resilient. The error tolerance in such applications may exist because of the redundancy or repetition in data, perpetual limits of the observer or presence of noise. The existence of this error tolerance in such applications, which usually handle a very large amount of data, mostly real-time, can be leveraged to relax the error bounds in arithmetic computations by using approximate computing and hence, optimizing the power consumption, latency (speed) and area of the digital processing unit (mainly composed of adders and multipliers).

Approximate computing can be applied to a particular application using various approaches. Numerous architectural, circuit and software level approximations [2] have already been proposed in the literature. For instance, in complex algorithms e.g., Artificial Neural Networks (ANN) [3], architectural-level optimizations/approximations can be applied, by identifying critical neurons. Circuit level approximations can be achieved by altering/simplifying logic circuits, like in Systematic Logic Synthesis of Approximate Circuits (SALSA) [4, 5], thus, optimizing the performance and power of the design at the cost of an erroneous output within certain bounds. Similarly, another approach for circuit level approximation is to design arithmetic datapaths with approximate adders [6–9] and approximate multipliers [10–13]. Also, software level approximations can be achieved using techniques like code perforation [2], etc. The selection for the right approach depends on the type of application, error resilience bounds and kind of optimization required.

In this chapter, we first explain component-level approximations and their probabilistic behavior by considering approximate adders and multipliers. Next, some methods used to construct efficient accelerators from these components is discussed. The discussion is then be extended to cross-layer approximations. Figure 1 explains the methodology flow of this chapter.

2 Component-Level Approximations: Adders and Multipliers

Adders and Multipliers are foundational blocks in all arithmetic circuits, including customized on-chip accelerators and general purpose processors. There exist a large number of computationally intensive applications which require a very high count of adders and multiplies. Many of these resource intensive applications do possess error resilience. Owing to this fact, the design of approximate adders and multipliers has attracted a great attention. Here, we give a brief overview of the approximate adders and multipliers from the existing literature.

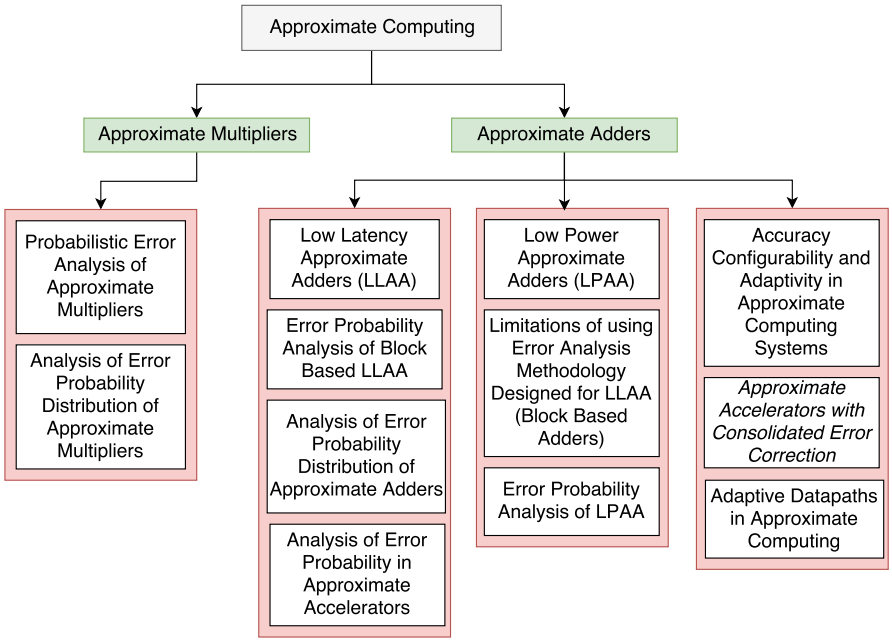


Fig. 1 Approximate Computing - Chapter Flow Diagram.

2.1 An Overview of Approximate Adders and Multipliers

There are several approaches used to design approximate adders and multipliers. Some common methods are described below:

- Some generic synthesis algorithms, SALSA [4] and IMPACT [14], have been used to design adders and multipliers. In this method, the Register Transfer Level (RTL) design of a precise circuit is input to the algorithm, which modifies it according to the design and performance constraints to give the approximate circuit as output.
- In [11, 13, 15, 16], approximate modules for the circuit building blocks are designed and then used to construct larger circuits. For example, in [11, 13, 15], approximate full adders (FAs) and approximate 2×2 multipliers are designed and then used to construct Ripple Carry Adders (RCA) and larger recursive multipliers. Similarly, in [13, 16], approximate $4:2$ compressors are used in Wallace tree multipliers for implementing the adder tree.
- In block-based adders [6,7,17,18], the adder inputs are divided among multiple disjoint or overlapping subadders. This type of design is inspired by the fact that the long carry propagations are relatively rare. Therefore, truncating the error chain introduces very infrequent errors.

- Many designs simplify the logic for less significant bits while keeping the higher significance bits accurate. In this way, the error magnitude is limited to relatively small values [19].
- Another technique (ABACUS) is to work out an optimal approximation design by applying iterative stochastic approach on abstract synthesis tree obtained from input behavioral description [20].

Different types of approximations discussed above have different types of error patterns, area, speed and power properties. For example, in case of low-power RCAs constructed from FAs, errors are more frequent and also distributed over a wide range of values. In contrast, errors in block-based adders are less frequent and can only have largely spaced values from a small set [21]. However, these are high-speed adders consume larger silicon area than precise adders.

3 Probabilistic Error Analysis

In traditional arithmetic units, the most commonly considered performance parameters are power consumption, critical path, latency (input to output delay) and area (silicon fabrication). However, approximate computing has introduced the computational accuracy as another design metric in any arithmetic unit. As in every application, error tolerance exists in the form of bounds, beyond which the result/output becomes useless. While designing an approximate arithmetic unit for an application, it is very important to evaluate and qualify for the accuracy metric along with other parameters, for achieving the maximum and reliable optimization. Accuracy of approximate adders and multipliers are statistically evaluated for maximum error/corruption.

An approximate computational system is considered as a deterministic system, i.e., based on a given input, the output can be reliably pre-evaluated.[1] Thus, the probabilistic analysis of approximate computing system provides the extent of corruption of random output produced because of a random input to a deterministic (approximate computing) system.

3.1 Probabilistic vs. Statistical Analysis

Conventionally, Monte-Carlo simulations were considered as the only and the most reliable method for the statistical analysis of approximate computing units to estimate error performance and provide comparisons between different approaches. In Monte-Carlo simulations, each configuration of approximate adder/multiplier needs to be simulated individually, which requires a considerable effort in reprogramming

[1] Another class of inexact computing is probabilistic computing, in which probabilistic switches are used, so that, in addition to the random inputs, the circuit's function is also random [22].

the simulation algorithm each time. The processing requirements and computation time are also quite high. Also, with Monte-Carlo simulations, it is not possible to locate the cause of error in terms of input distributions and other approximation parameters [21]. Exhaustive simulations, in which a system under consideration (in our case, we have an approximate computing unit) is tested for every possible input value while comparing its result to the correct one, is only a feasible solution for small circuits [21]. However, when the approximate computing unit contains a large number of stages/components, an exponential increase in the simulation time and number of arithmetic computations can be observed as shown in Fig. 2 [23].

In comparison to above, probabilistic analysis of approximate adders and multipliers can yield mathematical models. On the basis of these models, the cause and other characteristics of error, and its dependency on input distributions and circuit parameters, can be explained. Thus, these probabilistic models can serve as a tool for improvement in approximation approaches. Furthermore, these probabilistic methods are scalable and generic, in contrast to the traditional techniques.

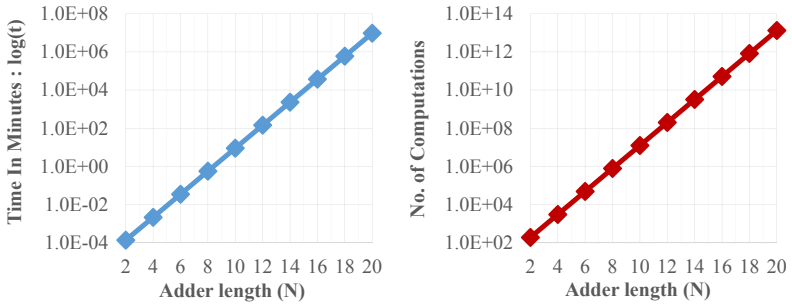


Fig. 2 Increase in Exhaustive Simulation Time and Number of Computations (Addtions, Comparisons etc.) vs. Adder length (N -bit) for a Circuit Based Approximation Approach (Intel Core i7 3rd Generation) [23].

3.2 Accuracy Metrics

Multiple performance parameters/metrics are used in the literature for quantifying the accuracy characteristics of approximate adders and multipliers. These parameters, after evaluation, are used for the qualification of approximate computing units in the application under consideration. Some of these performance metrics in the domain of approximate computing include minimum acceptable accuracy (MAA), accuracy of amplitude (ACCamp), accuracy of information (ACCinf), mean error distance (MED), normalized error distance (NED), error rate (ER), error signifi-

cance (ES) and maximum error magnitude [22, 24–26]. Traditionally, Monte-Carlo simulations is a widely known technique for the estimation of these parameters.

In modern literature, accuracy characteristics of approximate computing units can be authentically evaluated using the probability mass function (PMF) of error (signed or absolute) value. PMF of the error value can be used to observe respective error magnitudes and all other aforementioned metrics. In the upcoming sections, modern probabilistic methodologies for analyzing the ER and PMF is discussed for a set of selected configurations and classes of adders and multipliers.

3.3 Probabilistic Error Analysis of Approximate Adders

Depending on the approximation techniques, adders can be optimized with respect to latency (e.g. GeAr [6]) and/or power [9, 14]. In this section, we explain the analysis methodologies developed in [21] and [23] for low latency approximate adders (LLAA) and low power approximate adders (LPAA), respectively.

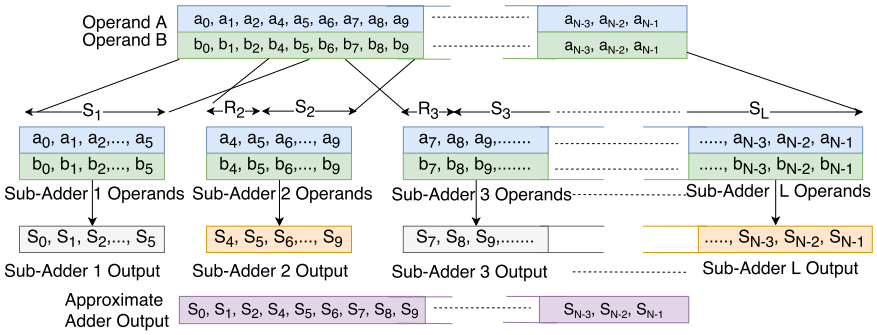


Fig. 3 Block-Based Adder Model.

3.3.1 Adder Model 1 (LLAA)

Fig. 3 shows a general model representing the block-based adders. The input bits are divided among multiple disjoint or overlapping subadders. Every subadder yields sum bits for the output from the corresponding input bits, while using some previous input bits for generating the carry-in. It should be noted here that Fig. 3 shows a *functional model* of the adder. Hardware implementation may differ depending upon the implementation platform. For instance, in [27], a more area-efficient implementation was used in which only the carry generation logic was implemented instead of the whole adder.

Analysis of Error Probability

It can be seen from Fig. 3 that the adder output is obtained by gathering the outputs of all the subadders. Therefore, the probability of error in the adder output is as follows:

$$\Pr[E] = \Pr[E_2 \vee ... \vee E_L], \tag{1}$$

where E_i is a binary random variable, such that $E_i = 1$ when the output of the i^{th} subadder is erroneous, and $E_i = 0$ otherwise. It can be seen from the model that first sub-adder's output is always accurate. An expanded form of this $\Pr[E]$ is found using the inclusion-exclusion principle as follows:

$$\begin{aligned}\Pr[E] = \sum_{i=2}^{L} \Pr[E_i] - \sum_{2\le i<j\le L} \Pr[E_i \wedge E_j] + \sum_{2\le i<j<k\le L} \Pr[E_i \wedge E_j \wedge E_k] - ... \\ +(-1)^L \Pr\left[\bigwedge_{i=2}^{L} E_i\right]\end{aligned} \tag{2}$$

Now, every intersection term in the above equation will be evaluated individually. Considering any i^{th} subadder, where $2 \le i \le L$, $E_i = 1$ when the following two events happen simultaneously:

1. The previous bits that are being used to generate the carry-in are all propagating, i.e., XOR of all input bit pairs is 1. The probability of this event with N propagating bits is

$$\Pr[P_i; N] = \Pr\left[\bigwedge_{n=1}^{N} A_n \oplus B_n = 1\right] = \frac{1}{2^N} \tag{3}$$

   where A_n, B_n is the n^{th} bit pair conributing to carry generation in the i^{th} subadder.
2. The previous less significant bits, that are not input to this i^{th} subadder are generating a carry-out. The probability of this event for K bit pairs is

$$\Pr[G_i; K] = \Pr[A_{1-K} + B_{1-K} \ge 2^K] = \frac{1}{2} - \frac{1}{2^{K+1}} \tag{4}$$

   where $A_{1-K} + B_{1-K}$ are all the less significant bits that are not input to i^{th} subadder

Thus $E_i = P_i \wedge G_i$. In case of intersection terms in (2), there are multiple such conditions. As a result, some groups of bits satisfy multiple conditions simultaneously and these conditions cannot be treated as independent. A simplification method in [21] was proposed to convert a set of these dependent conditions into independent ones, so that the probabilities of intersection can be represented as product of probabilities of individual conditions. This method is explained through the following example.
Example: Consider an adder with 3 subadders, as shown in Fig. 4. For ease of dis-

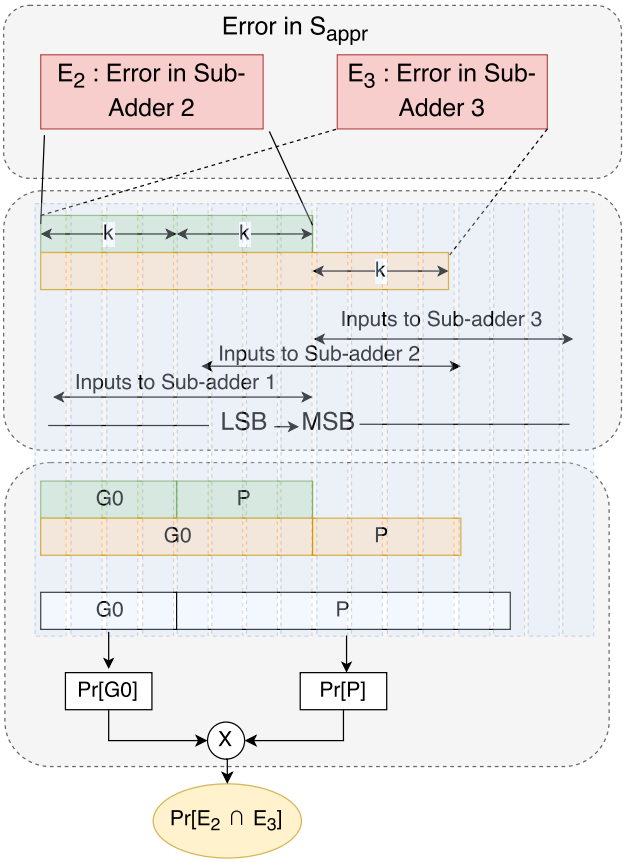


Fig. 4 Example of an Adder with 3 Subadders for the Derivation of $Pr[E_2 \wedge E_3]$ [21].

cussion, uniformly distributed inputs are assumed. The probability of error is given as:

$$\begin{aligned} \Pr[E] &= \Pr[E_2 \vee E_3] = \Pr[E_1] + \Pr[E_3] - \Pr[E_2 \wedge E_3] \\ &= \Pr[P_1 \wedge G_1] + \Pr[P_2 \wedge G_2] - \Pr[P_1 \wedge G_1 \wedge P_2 \wedge G_2] \\ &= \Pr[P_1]\Pr[G_1] + \Pr[P_2]\Pr[G_2] - \Pr[P_1 \wedge G_1 \wedge P_2 \wedge G_2] \end{aligned} \tag{5}$$

For the terms of the form $\Pr[P_i \wedge G_i]$, the two variables P_i and G_i are defined for disjoint groups of bits, as shown in Fig. 4. For uniform distribution, all input bits are identically distributed, so P_i and G_i are independent and hence $\Pr[P_i \wedge G_i] = \Pr[P_i]\Pr[G_i]$. For the term $Pr[P_1 \wedge G_1 \wedge P_2 \wedge G_2]$, some conditions are defined for same bits. For example, some bits are common for P_1, G_1 and G_2. This set of dependent conditions is converted into another set of independent conditions using the method in [21]. The conversion is shown in Fig. 4b. The $\Pr[E]$ is found as follows:

$$\Pr[E] = \Pr[P_1]\Pr[G_1] + \Pr[P_2]\Pr[G_2] - \Pr[G_1]\Pr[P_1]\Pr[P_2] \tag{6}$$

All the probabilities in the above equation can be found using (3) and (4).

Analysis of Error Probability Distribution

In order to find the PMF of error, the probabilities of all possible values of error need to be evaluated. The error in any one subadder has a fixed value determined by the location of carry chain truncation. For the example in Fig. 4, subadder 3 can only have an error with magnitude 2^{3k}. This error happens when there subadder 3 is erroneous while the output of subadder 2 is accurate. Thus, for every value of adder, the expressions of the form $\Pr[E_2 \wedge E_3 \wedge \overline{E_4} \wedge \overline{E_5}]$ need to be evaluated. This is done as follows:

$$\begin{aligned} &\Pr[E_2 \wedge E_3] = \Pr[E_2 \wedge E_3 \wedge (\overline{E_4} \wedge \overline{E_5})] + \Pr[E_2 \wedge E_3 \wedge (\overline{\overline{E_4} \wedge \overline{E_5}})] \\ &\implies \Pr[E_2 \wedge E_3 \wedge (\overline{E_4} \wedge \overline{E_5})] = \Pr[E_2 \wedge E_3] - \Pr[E_2 \wedge E_3 \wedge (E_4 \vee E_5)] \end{aligned} \tag{7}$$

where $\Pr[E_2 \wedge E_3 \wedge (E_4 \vee E_5)] = \Pr[E_2 \wedge E_3 \wedge E_4] + \Pr[E_2 \wedge E_3 \wedge E_5] - \Pr[E_2 \wedge E_3 \wedge E_4 \wedge E_5]$, which contains only joint probabilities that can be evaluated using the method in previous section. In [21], an algorithm is presented to find the probabilities of all possible combinations of errors. Further details relating to the nature of overlap among subadders need to be considered for an accurate analysis.

3.3.2 Adder Model 2 (LPAA)

A multi-bit LPAA is composed of cascaded single bit LPAA in configurations, like ripple carry adder (RCA), carry save adder (CSA), carry lookahead adder (CLA), etc. A single bit LPAA is usually developed using circuit level approximation approaches as discussed earlier. Though, the power consumption difference for a single bit LPAA is quite small in comparison to a single-bit accurate full adder (AccuFA). However, this power saving becomes significant when a large number of these low power adder units are used in an arithmetic logic unit. Table 1 discusses various single-bit LPAA models present in literature [9] [28]. Bold red outputs in the truth table signifies the errors caused by these LPAA (either in sum and/or carry).

Analysis of Error Probability

The major challenge in analyzing the multi-bit LPAA using the Principle of Inclusion-Exclusion technique discussed for LLAA model is the number of stages which are significantly high in the LPAA adder model. This is because of the reason that in multi-bit LPAA, each bit of adder is considered as a separate stage or block unlike LLAA adder model. Thus, an exponential expansion occurs in the number of equation terms in Principle of Inclusion-Exclusion which makes the solution highly computationally extensive [23]. In order to overcome the mentioned challenge, an

Table 1 Truth Tables of Different Single Bit LPAA as Proposed in [9] [28].

Inputs			**AccuFA**		**LPAA 1**		**LPAA 2**		**LPAA 3**		**LPAA 4**		**LPAA 5**		**LPAA 6**		**LPAA 7**	
A	B	Cin	Sum	C_{out}	Sum	C_{out}	Sum	C_{out}	Sum	C_{out}	Sum	C_{out}	Sum	C_{out}	Sum	C_{out}	Sum	C_{out}
0	0	0	0	0	0	0	**1**	**0**	**1**	**0**	0	0	0	0	0	0	0	0
0	0	1	1	0	1	0	1	0	1	0	1	0	**0**	**0**	**1**	**1**	1	0
0	1	0	1	0	**0**	**1**	1	0	**0**	**1**	**0**	**0**	1	0	1	0	1	0
0	1	1	0	1	0	1	0	1	0	1	**1**	**0**	**1**	**0**	0	1	**1**	**1**
1	0	0	1	0	**0**	**0**	1	0	1	0	**0**	**1**	**0**	**1**	1	0	1	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	**1**	**1**
1	1	0	0	1	0	1	0	1	0	1	0	1	**1**	**1**	**0**	**0**	0	1
1	1	1	1	1	1	1	**0**	**1**	**0**	**1**	1	1	1	1	1	1	1	1

iterative approach was presented in [23], which has a significantly low computational requirement, is linearly scalable and generic for present and future single-bit LPAA designs. This iterative technique provides the probability of error for multi-bit LPAA with foreknown input distribution.

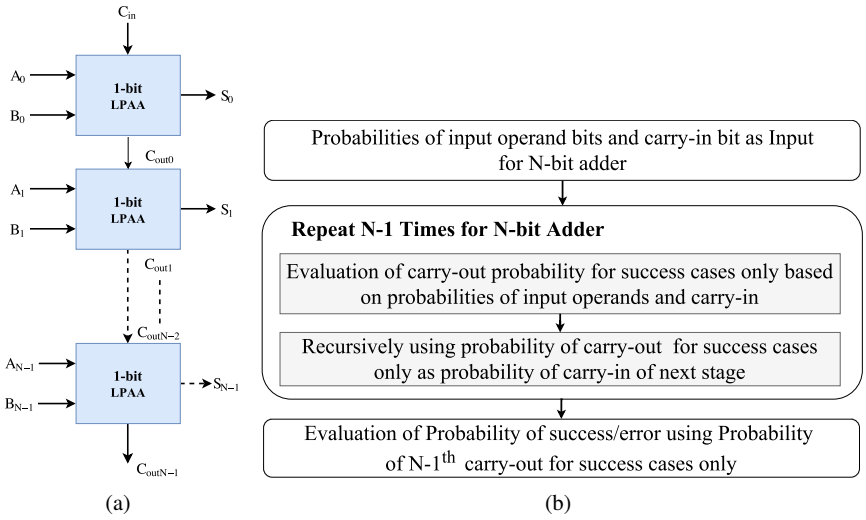


Fig. 5 (a). Block Diagram of Multi-bit Adder, (b). Block Level Implementation of LPAA Error Analysis Methodology [23].

Consider a multi-bit LPAA in RCA configuration as shown in Fig. 5a with operands A, B and C_{in}. It can be observed that in any adder configuration, carry propagates through the length of the adder. A corruption in carry-out or sum can cause a corruption in the result produced by the adder. The proposed algorithm works on the basic rule of statistically discarding the probability of erroneous carry-out from previous stage to the next stage and thus, each next stage considers the

probability of carry for the success cases only. The probability of carry for success cases only decreases along the length of adder because of the corruption caused by each approximate adder stage. Fig. 5b shows the block level implementation of methodology.

As shown in Fig. 5b, the first step in the methodology is to estimate the carry-out bit probability for the success cases only, using the probabilities of input operands (A, B) and probability of carry-in. The probability of carry-out for successful cases, estimated in the current stage, is used in the next iteration for the later stage. Thus, for an $N-bit$ adder (stages), this process is repeated until $N-1^{th}$ stage. At the last stage, the probability of success for the complete multi-bit adder, can be evaluated using previous stage ($N-2^{th}$) carry for success cases only and last stage input operand probabilities. The probability of error for the multi-bit adder can then be obtained directly from the probability of success. The same methodology can be used for probabilistic error analysis of accelerator configurations as well. Like other probabilistic error analysis techniques, this methodology also assumes that input operands are statistically independent.

To understand the methodology mathematically, consider operands A and B such that $A_0, A_1, ..., A_{N-1}$ and $B_0, B_1, ..., B_{N-1}$ and the carry bit C_{in}, with corresponding probabilities $P(A_i), P(B_i), P(C_{in})$ where $i = 0, 1, .., N-1$ for an N-bit adder. Also, it can be observed from the truth table for LPAA 1 that the probability of carry-out to be '1' or '0' for the success cases only changes in an inhomogeneous fashion and unlike other probabilistic scenarios, a conjugate relation cannot work for these two probabilities. Therefore, the methodology estimates the probability of carry-out to be '1' (true) for success cases separately with probability of carryout to be '0' (false) for success cases only. The terminologies given below, are used in the mathematical interpretation of the methodology.

- $P(C_{curr}) = P(C_{in})$ to be "1" at the current stage
- $P(\overline{C_{curr}}) = P(C_{in})$ to be "0" at current the stage
- $P(C_{next}) = P(C_{out})$ to be "1" at the current stage
- $P(\overline{C_{next}}) = P(C_{out})$ to be "0" at the current stage
- $P(Succ)$ = Probability of Success
- $P(\overline{Succ})$ = Probability of No Success

The generic methodology for error analysis of LPAA uses 3 matrices (M, K, L). These matrices can be evaluated using truth table for single-bit LPAA (Table 1) as below.

1. **M-matrix** : 1x8 matrix $M = [m_1, m_2, ..., m_8]$, such that $m_i = 1$ if C_{out} is "1" *AND* the case is a *Success* else $m_i = 0$.
2. **K-Matrix** : 1x8 matrix $K = [k_1, k_2, ..., k_8]$, such that $k_i = 1$ if C_{out} is "0" *AND* the case is a *Success* else $k_i = 0$
3. **L-Matrix**: 1x8 Matrix $L = [l_1, l_2, ..., l_8]$, such that $l_i = 1$ if the case is a *Success* else $l_i = 0$. The above defined matrices remain the same for any LPAA type and are irrespective of length or configuration of adder. *M, K* and *L* matrices for various LPAA topologies are given in Table 2.

4. *Input probability matrix (IPM)* estimates the probability of occurrence of each state of the truth table and can be iteratively evaluated as below:

$$\begin{aligned} IPM = [&P(\overline{A_i}).P(\overline{B_i}).P(\overline{C_{curr}} \cap Succ), P(\overline{A_i}).P(\overline{B_i}).P(C_{curr} \cap Succ), \\ &P(\overline{A_i}).P(B_i).P(\overline{C_{curr}} \cap Succ), P(\overline{A_i}).P(B_i).P(C_{curr} \cap Succ), \\ &P(A_i).P(\overline{B_i}).P(\overline{C_{curr}} \cap Succ), P(A_i).P(\overline{B_i}).P(C_{curr} \cap Succ), \\ &P(A_i).P(B_i).P(\overline{C_{curr}} \cap Succ), P(A_i).P(B_i).P(C_{curr} \cap Succ)] \end{aligned} \tag{8}$$

5. Using the IPM, along with M and K matrices, we can find the probability of the carry-out signal using a simple dot product:

$$\begin{aligned} P(C_{next} \cap Succ) &= [IPM].[M] \\ P(\overline{C_{next}} \cap Succ) &= [IPM].[K] \end{aligned} \tag{9}$$

6. Steps 4 and 5 are repeated $N-1$ times in case of an N-bit adder. For every iteration, we use the evaluated carry probabilities from current stage to evaluate the ones for the next stage.
7. After $N-1$ iterations, the probability of success and error can be evaluated by dot product as follow:

$$\begin{aligned} P(Succ) &= [IPM].[L] \\ P(Error) &= 1 - P(Succ) \end{aligned} \tag{10}$$

Table 2 *M, K* and *L* Matrices Required for Analysis of LPAA 1-7 Proposed in [28] [9].

Matrix / LPAA Type	LPAA 1	LPAA 2	LPAA 3	LPAA 4	LPAA 5	LPAA 6	LPAA 7
M-Matrix	[0,0,0,1,0,1,1,1]	[0,0,0,1,0,1,1,0]	[0,0,0,1,0,1,1,0]	[0,0,0,0,0,1,1,1]	[0,0,0,0,0,1,0,1]	[0,0,0,1,0,1,0,1]	[0,0,0,0,0,0,1,1]
K-Matrix	[1,1,0,0,0,0,0,0]	[0,1,1,0,1,0,0,0]	[0,1,0,0,1,0,0,0]	[1,1,0,0,0,0,0,0]	[1,0,1,0,0,0,0,0]	[1,0,1,0,1,0,0,0]	[1,1,1,0,1,0,0,0]
L-Matrix	[1,1,0,1,0,1,1,1]	[0,1,1,1,1,1,1,0]	[0,1,0,1,1,1,1,0]	[1,1,0,0,0,1,1,1]	[1,0,1,0,0,1,0,1]	[1,0,1,1,1,1,0,1]	[1,1,1,0,1,0,1,1]

This methodology has insignificant computational requirements for any adder length and it can cater-for any LPAA design as it needs just three matrices defining the behavior of the adder to work. These three matrices can be easily evaluated from the truth table of the new LPAA as defined above. The same methodology can also be applied for probability of error analysis of LLAA (Adder Model 1).

3.4 Probabilistic Error Analysis of Approximate Multipliers

3.4.1 Multiplier Model

Fig. 6 shows a general model of the recursive approximate multipliers [11, 15, 29]. An $N \times N$ multiplier is constructed from $M \times M$ approximate multiplier building blocks. Several low-power 2×2 approximate multipliers have been designed [2, 11,

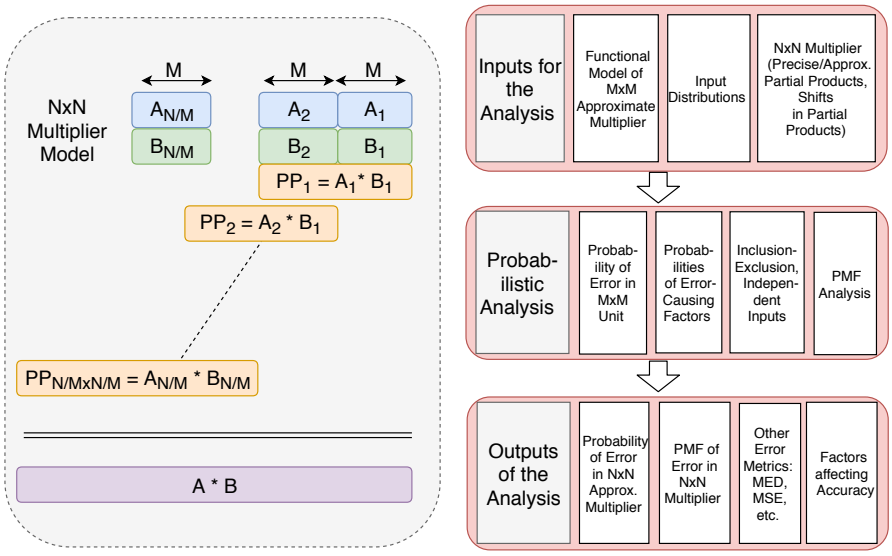


Fig. 6 Model of Recursive Approximate Multiplier.

15] for this type of multipliers. Another approach used is to use 4×4 components, which employ approximate $4:2$ compressors [13, 30].

3.4.2 Analysis of Error Probability

For a given $N \times N$ multiplier employing $M \times M$ components, the PMF of error in $M \times M$ component is known from its truth table. Let Θ be the set containing all M-bit values of inputs that lead to error in the output of $M \times M$ multipliers. The probability of error in $M \times M$ multiplier with M-bit inputs X_M and Y_M is defined as $\rho_M = \Pr[X_M \in \Theta \wedge Y_M \in \Theta]$. In Fig. 6, since every M-bit component of input A is multiplied with every M-bit component of input B, the $\Pr[E]$ is found as follows:

$$\Pr[E] = \Pr\left[\left(\bigvee_{i=1}^{N/M} A_i \in \Theta\right) \wedge \left(\bigvee_{i=1}^{N/M} B_i \in \Theta\right)\right] \tag{11}$$

where A_i and B_i are groups of M bits of inputs A and B, respectively. For uniformly distributed inputs, A_i and B_i are independent for all $i \in \{1, 2, ..., N/M\}$. Moreover, the two inputs A and B are also assumed to be independent and identically distibuted. Therefore, $\Pr[A_i \in \Theta] = \Pr[B_i \in \Theta] = \sqrt{(\rho_M)}$. Hence,

$$\Pr[E] = \Pr\left[\bigvee_{i=1}^{N/M} A_i \in \Theta\right] \Pr\left[\bigvee_{i=1}^{N/M} B_i \in \Theta\right] = \left(\Pr\left[\bigvee_{i=1}^{N/M} A_i \in \Theta\right]\right)^2 \tag{12}$$

Table 3 PMF Analysis for 4×4 multiplier employing Kulkarni's 2×2 design.

Input conditions	Probability	Erroneous Partial Products	Error Magnitude
$A_1, B_1 = 11_2$	1/16	A_1B_1	2
$A_2, B_1 = 11_2$	1/16	A_2B_1	8
$A_1, A_2, B_1 = 11_2$	1/64	A_1B_1, A_2B_1	$2+8=10$
$A_1, B_2 = 11_2$	1/16	A_1B_2	8
$A_2, B_2 = 11_2$	1/16	A_2B_2	32
$A_1, A_2, B_2 = 11_2$	1/64	A_1B_2, A_2B_2	$8+32=40$
$A_1, B_1, B_2 = 11_2$	1/64	A_1B_1, A_1B_2	$2+8=10$
$A_2, B_1, B_2 = 11_2$	1/64	A_2B_1, A_2B_2	$8+32=40$
$A_1, A_2, B_1, B_2 = 11_2$	1/256	$A_1B_1, A_1B_2, A_2B_1, A_2B_2$	$2+8+8+32=50$

The probability above can be evaluated using the inclusion-exclusion principle and due to the independence property of A_i and B_i, for $i \in \{1,2,...,N/M\}$, the intersection terms in the inclusion-exclusion expansion can be represented as products of $\sqrt{\rho_M}$. Hence, the expression simplifies as follows:

$$\Pr[E] = \left(\sum_{i=1}^{N/M} (-1)^{i+1} \binom{N/M}{i} (\sqrt{\rho_M})^i \right)^2 \tag{13}$$

The method described above is exact for those multipliers that match with the assumptions in their behavior [11, 30]. For other multipliers [2, 13], it gives a good approximation [29].

3.4.3 Analysis of Error Probability Distribution

For PMF, all possible values of error and their probabilities need to be evaluated. In [29], an algorithm was developed for this purpose. This algorithm considers all possible combinations of M-bit groups of each input and their corresponding probabilities. Error value is determined by identifying which partial products will be erroneous and multiplying the error in $M \times M$ component with the power of shift of these partial products. The evaulation of error probability and PMF is explained through the following example.

Example: Consider an 4×4 multiplier made up of 4 Kulkarni's multipliers which are 2×2 components. Kulkarni's multiplier introduces an error of magnitude 2 when both inputs are 11_2, so $\rho_M = 1/16$ and $\Theta = \{11_2\}$. Probability of error is found as follows:

$$\begin{aligned} \Pr[E] &= \left(\Pr[A_1 \in \Theta] + \Pr[A_2 \in \Theta] - \Pr[A_1 \in \Theta \wedge A_2 \in \Theta]\right)^2 \\ &= \left(\sqrt{\rho_M} + \sqrt{\rho_M} - (\sqrt{\rho_M})^2\right)^2 = 0.1914 \end{aligned} \tag{14}$$

For the PMF, all possible errors are considered individually and their probabilities are determined. Table 3 shows the analysis. For larger multipliers and $M \times M$ multipliers with a more complicated behavioral model, Algorithm 1 in [29] is used to automate this analysis.

3.5 Analysis with Arbitrary Input Probability Distributions

3.5.1 Probability of Error in Accelerators

If there are more than one approximate component in a circuit, such that the cumulative error is the sum of errors in individual components, an approximated probabilistic analysis can be carried out by assuming that all the components are independent and all the intermediary inputs are approximately uniformly distributed. The PMF of the cumulative error value is defined as the convolution of PMFs of individual components. This method was shown to yield good approximation in [21]. Similarly, discrete cosine transform (DCT) based JPEG multimedia compression using approximate accelerators was designed in [31]. This approximate design is proven to save up to 15% area on silicon chip and 40% reduction in power consumption.

4 Accuracy Configurability and Adaptivity in Approximate Computing Systems

Considering the difference in the demand of accuracy in various applications and also taking into account the probable alterations in the configuration of approximate module during application runtime, many multipliers and adders have been designed to include integrated error detection and correction (EDC) units [6, 7, 11, 30]. Nevertheless, EDC introduces an overhead, which expands with size of approximate processing unit.

4.1 Approximate Accelerators with Consolidated Error Correction

As previously discussed, approximate adder and multipliers are equipped with EDC to benefit in certain environments. However, when the size of datapath increases, the overhead introduced by EDC becomes non-trivial and benefits of having approximate computing get suppressed. Recently, a consolidated error correction (CEC) approach was proposed to overcome this limitation [32]. The reduction in overhead was achieved by introducing Error Detection (ED) for all components in approximate computing unit. The generated ED signals are then used for joint CEC for a group of components. Grouping the larger number of components together in the same manner results in further optimization of area and speed of approximate computing unit with error correction.

Example: Fig. 7 shows an accelerator, composed of block-based adder (model 1), with a common CEC implemented for a group of 8 adders. This simplification in the circuit design is due to the fact that, in block-based adder circuits, error can attain certain specific values, which depend on the location from where carry-chain

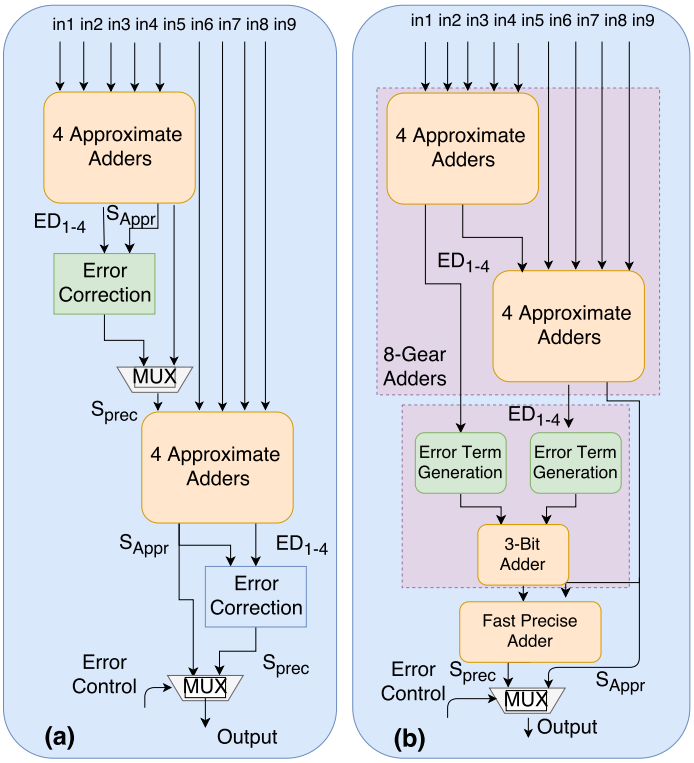


Fig. 7 Accelerator with Error Detection.

has been truncated. By locating these truncation positions, the ED signals can easily calculate the error value. In terms of speed and area optimizations, this design is found to be more efficient than the EDC designs in [6, 7].

4.2 Adaptive Datapaths

The major objective of the adaptive approximate computing [27] is to minimize the accumulated error by designing a datapath with several approximate additions and multiplications, such that the error in one operation, controls the approximate module used in the subsequent operation. To achieve this, *complementary* approximate modules were designed, such that the datapath is composed of both standard and complementary versions of the module. The complementary versions are such circuits that generate errors with opposite signs as compared to the standard versions. The selection of module in the subsequent operation depends on the ED signals.

Following components and control mechanisms are needed to design the datapath using the algorithm in Fig. 8.

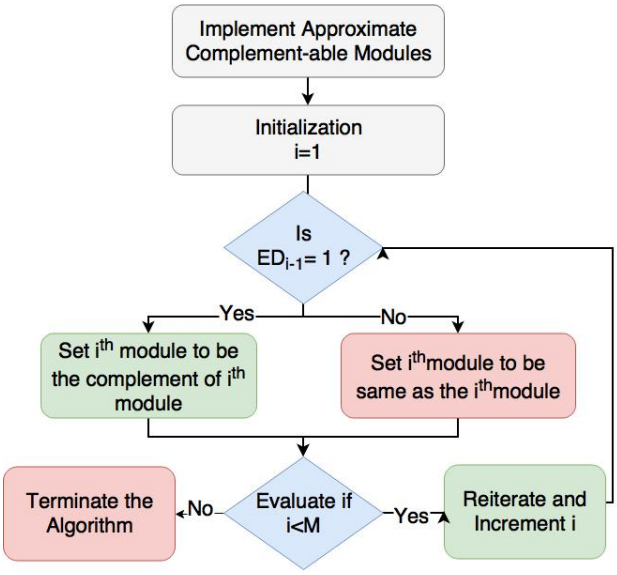
- Approximate modules with complementary and standard versions. Preferably, the complementary and standard versions of approximate module should have 1). opposing polarities, same error magnitudes and equal probabilities, i.e., p (ESAM)(x) = p (ECAM)(x) and, 2). same silicon area, power consumption and speed/performance.
- For every component in the datapath, an ED signal has to be generated.
- A signal or mechanism S/C which can indicate the type of module currently being used.
- Using the S/C and ED signals, a control logic or mechanism for switching between the complementary and standard modules

With the above explained datapath design, the error in any two modules gets partially or completely cancelled, causing a significant reduction in the cumulative error. If complementary and standard versions of approximate modules are capable of completely nullifying the error introduced by each other, the cumulative error shall be non-zero only when there exists non-even number of erroneous modules in the datapath. Also, in this case, the maximum error accumulates in the complete datapath shall be limited to the maximum error in a single approximate module. Thus, with this method, if datapath is designed using M approximate modules, a reduction of up to M times can be achieved in maximum cumulative error in comparison with conventional homogeneous accelerator.

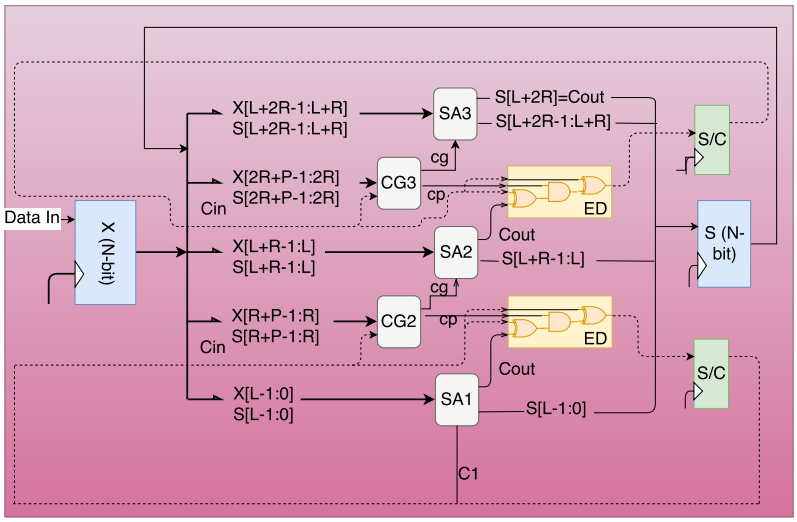
Example: For the better understanding of the adaptive datapath, consider the example of block-based adders working as an accumulator. To simplify the discussion, we consider an adder with two subadders as given in Fig. 8b. As prominent from the figure, *cin* signal is used to identify between complementary and standard modules. The ED signal will attain value 1 whenever the carry generated by the first subadder is different from the *cin* signal of the other subadder. As per 8a, the error in one addition enforces the next addition to be carried out by the complementary approximate module. In this way, the maximum error magnitude in this subadder remains confined to the error magnitude in one approximate adder [27].

5 Conclusion

Arithmetic logic units (ALU), containing adders and multipliers, can benefit from approximate computing due to the inherent error resilience in multiple applications. However, the error introduced in the computations because of approximate computing elements, should stay in particular bounds to avoid any malfunctioning. Therefore, based on the expected input operand characteristics, error in any approximate computing application needs to be pre-evaluated. In this chapter, we have discussed various approximation techniques and different metrics for the qualification of an approximate architecture in an arithmetic application. Also, we have discussed the statistical error analysis techniques for different variants of approximated adders, accelerators and multipliers. Lastly, we have explained multiple approaches for er-



(a)



(b)

Fig. 8 Block-Based LLAA as an Accelerator.

ror detection and correction in these approximation units along with the insight of adaptive datapaths.

References

1. Q. Xu, N. S. Kim, and T. Mytkowicz, "Approximate computing: A survey," *IEEE Des. & Test*, vol. 33, no. 1, pp. 8–22, 2016.
2. M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, "Cross-Layer Approximate Computing: From Logic to Architectures," in *Proc. 53rd IEEE/ACM Des. Autom. Conf.*, 2016.
3. Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "Approxann: an approximate computing framework for artificial neural network," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 701–706.
4. S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: systematic logic synthesis of approximate circuits," in *Proc. 49th IEEE/ACM Des. Autom. Conf.*, 2012, pp. 796–801.
5. A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan, "ASLAN: Synthesis of approximate sequential circuits," in *Proc. Des., Autom. Test Eur. Conf. Exhib*, 2014, p. 364.
6. M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *Proc. 52nd Annual Des. Autom. Conf.*, 2015, p. 86.
7. A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Proc. 49th Annual Des. Autom. Conf.*, 2012, pp. 820–825.
8. K. Du, P. Varman, and K. Mohanram, "High performance reliable variable latency carry select addition," in *Proc. Des., Autom. Test Eur. Conf. Exhib*, 2012, pp. 1257–1262.
9. V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Trans. Comput.-Aided Des. Integ. Circuits Syst.*, vol. 32, no. 1, pp. 124–137, 2013.
10. K. Bhardwaj and P. S. Mane, "Acma: Accuracy-configurable multiplier architecture for error-resilient system-on-chip," in *Proc. 8th Int. Workshop Reconfig. Commun.-Centric Syst.-on-Chip*, 2013, pp. 1–6.
11. P. Kulkarni, P. Gupta, and M. D. Ercegovac, "Trading Accuracy for Power in a Multiplier Architecture," *Journal of Low Power Electron.*, vol. 7, no. 4, pp. 490–501, 2011.
12. I.-C. Chen and J. P. Hayes, "Low-area and high-speed approximate matrix-vector multiplier," in *IEEE 18th Int. Symp. Des. Diagnostics of Electron. Circuits & Syst.*, 2015, pp. 23–28.
13. A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of approximate compressors for multiplication," *IEEE Trans. Comput.*, vol. 64, no. 4, pp. 984–994, 2015.
14. V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "IMPACT: imprecise adders for low-power approximate computing," in *Proc. 17th IEEE/ACM Int. Symp. Low-power Electron. & Des.*, 2011, pp. 409–414.
15. S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, and J. Henkel, "Architectural-Space Exploration of Approximate Multipliers," in *Proc. Int. Conf. Comput.-Aided Des.*, 2016, pp. 1–6.
16. J. Ma, K. Man, T. Krilavicius, S. Guan, and T. Jeong, "Implementation of high performance multipliers based on approximate compressor design," in *Proc. Int. Conf. Electrical and Control Technol.*, 2011.
17. 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. Integ. Circuits,*, 2009, pp. 69–72.
18. R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *Proc. Int. Conf. Comput.-Aided Des.*, 2013, pp. 48–54.
19. S. Hashemi, R. Bahar, and S. Reda, "Drum: A dynamic range unbiased multiplier for approximate applications," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press, 2015, pp. 418–425.

20. K. Nepal, Y. Li, R. Bahar, and S. Reda, "Abacus: A technique for automated behavioral synthesis of approximate computing circuits," in *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2014, p. 361.
21. S. Mazahir, O. Hasan, R. Hafiz, M. Shafique, and J. Henkel, "Probabilistic error modeling for approximate adders," *IEEE Transactions on Computers*, vol. 66, no. 3, pp. 515–530, 2017.
22. J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *IEEE Transactions on Computers*, vol. 62, no. 9, pp. 1760–1771, 2013.
23. M. K. Ayub, O. Hasan, and M. Shafique, "Statistical Error Analysis of Low Power Approximate Adders," in *Design Automation Conference*. ACM, 2017.
24. W.-T. J. Chan, A. Kahng, S. Kang, R. Kumar, and J. Sartori, "Statistical analysis and modeling for error composition in approximate computation circuits," in *Proc. IEEE 31st Int. Conf. Comput. Des.*, 2013, pp. 47–53.
25. R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "MACACO: Modeling and analysis of circuits for approximate computing," in *Proc. Int. Conf. Comput.-Aided Des.*, 2011, pp. 667–673.
26. J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *18th IEEE Eur. Test Symp.*, 2013, pp. 1–6.
27. S. Mazahir, O. Hasan, and M. Shafique, "Adaptive approximate computing in arithmetic datapaths," *IEEE Design & Test*, 2017.
28. H. A. Almurib, T. Kumar, and F. Lombardi, "Inexact Designs for Approximate Low Power Addition by Cell Replacement," in *Design, Automation & Test in Europe*. IEEE, 2016, pp. 660–665.
29. S. Mazahir, O. Hasan, R. Hafiz, and M. Shafique, "Probabilistic error analysis of approximate recursive multipliers," *IEEE Transactions on Computers*, vol. 66, no. 11, pp. 1982–1990, 2017.
30. C.-H. Lin and C. Lin, "High accuracy approximate multiplier with error correction," in *Proc. IEEE 31st Int. Conf. Comput. Des.*, 2013, pp. 33–38.
31. F. S. Snigdha, D. Sengupta, J. Hu, and S. S. Sapatnekar, "Optimal design of jpeg hardware under the approximate computing paradigm," in *Design Automation Conference*. ACM, 2016, pp. 106:1–106:6.
32. S. Mazahir, O. Hasan, R. Hafiz, M. Shafique, and J. Henkel, "An Area-Efficient Consolidated Configurable Error Correction for Approximate Hardware Accelerators," in *Proc. IEEE/ACM 53rd Des. Autom. Conf.*, 2016.