AxMAP: Making Approximate Adders Aware of Input Patterns

Morteza Rezaalipour, Mohammad Rezaalipour, Masoud Dehyadegari, and Mahdi Nazm Bojnordi

Abstract—Making approximate computing specific to user requirements is crucial to system performance, energy-efficiency, and reliability. However, developing hardware for such optimization becomes a significant challenge due to the high cost of examining all potential choices while exploring a large design space. One determinant aspect of exploring a design space is the efficiency of evaluating error metrics, such as the Mean Error Distance (MED) and the Error Probability (EP), for each possible choice within the search space. Since computing these error-metrics is quite time-consuming, efficient calculation approaches are essential.
This paper proposes a novel formal approach to accurately compute the EP and MED of approximate adders for any input pattern at a linear time and space complexity. Our experimental results indicate that the proposed approach can accurately compute error-metrics of large approximate adders at a 150 times faster speed compared to the Monte Carlo sampling methods. We then develop AxMAP, a design tool based on the proposed error-metrics computation that generates energy-efficient approximate adders for any given input pattern. When applied to image processing applications, AxMAP produces more than 150 different designs for adders that achieve superior performance and energy-efficiency compared to the existing state-of-the-art approximate adders.

Index Terms—Approximate Computing, Adders, Mean Error Distance, Error Probability, Circuit Synthesis

1 Introduction

Approximate computing trades accuracy for power, area, and delay of modern-day applications such as machine learning and image processing [1]. Digital adders are the key component of a wide range of error-resilient applications. In general, Approximate adders have become the main focus of numerous recent [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13]. Approximate adders have been classified to *Low Power Approximate Adder* (LPAA) [2], [5], [7], [8], [10], [11], [13], and *Low Latency Approximate Adder* (LLAA) [3], [4], [6], [9], [12].

Prior work has extensively used various error-metrics such as the Mean Error Distance (MED) and Error Probability (EP) to assess the error characteristics of approximate adders [14], [15], [16], [17]. Computing these metrics is quite time consuming as they often require obtaining the results of every possible input pattern, which is 2^{2n} instances for an n-bit adder. The time complexity of computing MED and EP for an n-bit adder is of the order of $O(2^{2n})$. However, an efficient and accurate computation of error-metrics is necessary to develop design automation tools that can synthesize approximate adders specific to user requirements [18], [19]. For instance Tajasob et al. [2] propose a vast design space with billions of billion approximate adders each having specific circuit and error characteristics. Exploring such huge space to find matches to users needs requires efficient methods of computing error-metrics.

Several analytical methods have been presented in the literature to compute error-metrics [1], [15], [20], [21], [22], [23]. Although they have been successful in addressing this problem partially, there are limitations for each. Some studies, such as the work of Liu et al. [15], avoid exhaustive and time consuming calculation of error metrics to some extent by employing Monte Carlo sampling methods. However, the results estimated by Monte Carlo methods are inexact. Moreover, these methods are time-consuming and largely impractical as the number of samples increases. Mazahir et al. [1] also mention limitations of Monte Carlo simulations for computing error characteristics of approximate adders.

Li and Zhou [23] propose a framework that computes MEDs of LLAAs accurately while the results produced for EPs are approximated. Their framework only works with uniform input patterns. The method presented by Liu et al. [15] estimates MED for different input patterns. However, this framework does not compute EP and employs different approaches for different types of adders. Mazahir et al. [1] propose to address the issue with a more general method for computing EPs and error distributions of LLAAs. It also employs error distributions to compute other metrics such as MED. This method works with different input patterns; however, it only produces the results for uniform input patterns accurately. Wu et al. [21] propose a method to compute EPs and MEDs of LLAAs which faster than the work by Mazahir et al. [1]. Both methods provide estimates of error-metrics for non-uniform input patterns.

It is worth stating that none of the studies mentioned above can calculate MED and EP for LPAAs. Ayub et al. [22] propose a method to obtain EPs of LPAAs for different input patterns. This method does not compute MED. Another method presented by Roy et al. [20] only computes MEDs of LPAAs, and does not work with different input patterns. Also, the complexity of the method presented in [20] is of the order of $O(2^m)$, where m is the length of the approximate portion.

In this paper, we present a novel formal approach that ac-

- *Morteza Rezaalipour and Masoud Dehyadegari are with the K. N. Toosi University of Technology.*
  E-mail: mrezaalipour@email.kntu.ac.ir and dehyadegari@kntu.ac.ir
- *Mohammad Rezaalipour is with the Software Institute, Faculty of Informatics, Università della Svizzera italiana (USI), Lugano, Switzerland.*
  E-mail: mohammad.rezaalipour@usi.ch
- *Mahdi Nazm Bojnordi is with the School of Computing, University of Utah, Salt Lake City.*
  E-mail: bojnordi@cs.utah.edu

Manuscript received April xx, 20xx; revised August xx, 20xx.

curately and efficiently computes EPs and MEDs for LPAAs, for different input patterns, in linear time and space complexities. This approach employs a model so-called the *quad-tree representation* [2] to provide generic MED and EP formulae that can be instantiated for any LPAA. The experimental results demonstrate the validity of our approach and indicate that it can compute MEDs, and EPs of large approximate adders more than 150 times faster than Monte Carlo sampling methods. We develop a tool called AxMAP that employs the proposed approach and automatically generate approximate adders. By exploring the design space, AxMAP finds a suitable adder for given input patterns, under specified circuit and error characteristics. Using this tool, more than 150 adders have been produced for an image processing problem each providing better trade-offs among power, area, delay, and MED compared to the existing state-of-the-art energy-efficient approximate adders. To show the generality of the results, we evaluate the generated adders over a wider range of input images.

2 Background and Observations

2.1 Preliminaries

The proposed approach in this paper is based on two primary concepts: fast and accurate computation of error-metrics and design space exploration using quad-tree representation [2]. This section provides an overview of these concepts.

2.1.1 Error-metrics

Liang et al. [16], [17] propose *Error Distance* (ED) as a metric to characterize the reliability of adders for a specific input pair. For a given n-bit approximate adder and two binary inputs $a = (a_{n-1}a_{n-2}...a_0)$ and $b = (b_{n-1}b_{n-2}...b_0)$, the error distance is defined as $ED_j = \hat{R_j} - R_j$, where R_j is the correct result (i.e, the result of an accurate adder for a and b), $\hat{R_j}$ is the value that the given approximate adder produces, and j is the index of the current input pair, which we compute as $j = (a_0b_0a_1b_1...a_{n-2}b_{n-2}a_{n-1}b_{n-1})_{10}$.

Based on the metric ED, two other well-known error-metrics are defined which are the *Mean Error Distance* (MED) and the *Error probability* (EP) [16]. EP represents the ratio of the number of input pairs whose ED is not zero, to the total number of input pairs (i.e., 2^{2n} for an n-bit adder). MED is the mean value of all EDs produced for an n-bit approximate adder, and it is shown in Eq. (1). The term Q_j represents the probability of input pair j, and for the uniform input pattern, since input pairs occur with the same probability (i.e., $1/2^{2n}$), Q_j can be replaced by $1/2^{2n}$.

$$MED = \sum_{j=0}^{2^{2n-1}} |ED_j|Q_j \tag{1}$$

2.1.2 Quad-tree Representation

The so-called quad-tree representation [2] is a generic form based on ED to model building blocks of a family of LPAAs known as disjoint approximate adders. The quad-tree representation has been employed to produce energy-efficient adders that demonstrate better trade-offs between circuit and error characteristics compared to the state-of-the-art adders. An n-bit disjoint approximate adder comprises a sequence of k-bit approximate sub-adders (i.e., disjoint building blocks) each of which produces a subset of the output bit positions. The disjoint building blocks in a sequence do not pass or receive carry signals, except for the last building block which produces the carry-out signal (C_{out}) of the final result. The state-of-the-art Lower-Part-OR adder (LOA) [13] is a disjoint approximate adder which is designed based on disjoint single-bit building blocks (i.e., $k = 1$).

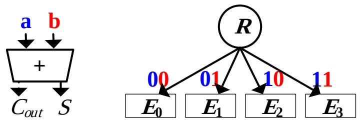


Fig. 1. Quad-tree representation of disjoint single-bit building blocks.

Figure 1 illustrates the quad-tree representation of disjoint single-bit building blocks. In this model, the node at level 0 is the root of the tree, which is labeled as R in Figure 1, and each node at the last level (e.g., the node labeled as E_0) is called *leaf*. A path starting from the root to a leaf demonstrates a specific input pair. Since disjoint single-bit building blocks do not receive carry-in signals (C_{in}), there are only four possible input pairs to them. Thus, there are four edges in their quad-tree representation, each having its corresponding input pair written on it in Figure 1. The parameter E_i ($0 \leq i \leq 3$) in each leaf, represents the ED computed for the input pair of the path to that leaf.

Since the last building block of each disjoint approximate adder produces a carry-out signal, it has a 2-bit wide output comprising the sum signal (S) and C_{out}. Thus, there are four possible values for each E_i depending on the exact output for its corresponding input pair. For instance, for input pair $(a, b = 0, 0)$, E_0 can be either of the values 0, +1, +2, and +3. As a result, there are 4^4 possible sequences of leaves, each of which represents a disjoint single-bit building block with the carry-out signal, regardless of hardware implementation.

The output of building blocks that do not have carry-out signals are 1-bit wide comprising only the sum signal (i.e., S). As a result, for these building blocks, there are only two possible values for each E_i. For example, for input pair $(a, b = 1, 1)$, E_3 can be either -1 or -2, which indicates the existence of 2^4 possible disjoint single-bit building blocks of this type.

The quad-tree representation is capable of modeling disjoint multi-bit building blocks as well (i.e., $k > 1$). However, due to space limitations, it is not reviewed here, and for further details, interested readers are directed to the article prepared by Tajasob et al. [2].

2.2 Observations

2.2.1 First Observation: MED Calculation

The primary purpose of the quad-tree representation is to model disjoint building blocks based on their EDs. However, after a thorough examination of the quad-tree representation, we realized that it could also represent disjoint approximate adders as a framework for analytical computation of error-metrics for approximate adders.

Figure 2 illustrates the quad-tree representation of an n-bit adder comprising disjoint single-bit building blocks. In this model, i ($i < n$) refers to the levels of the tree, and in each level, j ($j < 2^{2i}$) is the index of nodes. Considering $a = (a_{n-1}a_{n-2}...a_0)$ and $b = (b_{n-1}b_{n-2}...b_0)$ as the binary input numbers to the adder, j is computed as $j = (a_0b_0a_1b_1...a_{n-2}b_{n-2}a_{n-1}b_{n-1})_{10}$. The depth of this quad-tree is equal to the length of the adder, which

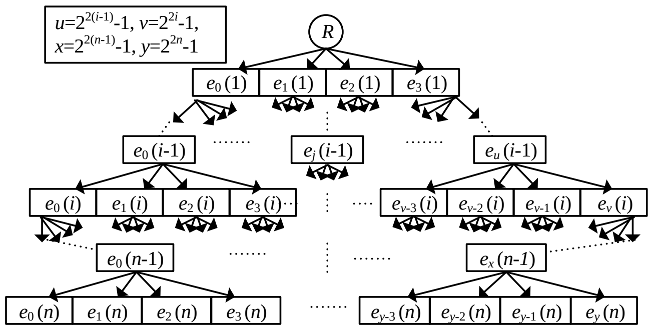

$u=2^{2(i-1)}-1$, $v=2^{2i}-1$,
$x=2^{2(n-1)}-1$, $y=2^{2n}-1$
R
$e_0(1)$ $e_1(1)$ $e_2(1)$ $e_3(1)$
$e_0(i-1)$ $e_j(i-1)$ $e_u(i-1)$
$e_0(i)$ $e_1(i)$ $e_2(i)$ $e_3(i)$
$e_{v-3}(i)$ $e_{v-2}(i)$ $e_{v-1}(i)$ $e_v(i)$
$e_0(n-1)$ $e_x(n-1)$
$e_0(n)$ $e_1(n)$ $e_2(n)$ $e_3(n)$
$e_{y-3}(n)$ $e_{y-2}(n)$ $e_{y-1}(n)$ $e_y(n)$

Fig. 2. Quad-tree representation of an n-bit disjoint approximate adder comprising single-bit building blocks.

is n. Node j at level i, referred to as $e_j(i)$, contains the ED of an i-bit sub-adder comprising the first i least significant bits (i.e., from bit-position 0 to i-1). Thus, for adders comprising disjoint single-bit building blocks, $e_0(1)$ to $e_3(1)$ are equal to E_0 to E_3. An edge starting from a parent node at level $i-1$ to one of its children at level i represents a_ib_i, the i^{th} bits of the two binary input numbers a and b.

Based on the presented model in Figure 2, each leaf at level n contains the ED value for one of the possible input pairs to the n-bit adder. As a result, for uniformly distributed input patterns, the MED of the family of adders represented in this figure can be computed by Eq. (2).

$$MED(n) = \frac{|e_0(n)| + |e_1(n)| + ... + |e_{2^{2n}-1}(n)|}{2^{2n}} \tag{2}$$

Since there are 2^{2n} addition operations in Eq. (2), its time complexity is of the order of $O(2^{2n})$, which needs to be reduced. On the other hand, considering the characteristics of the quad-tree in Figure 2, the value in each four siblings at level i can be expressed based the value of their parents at level $i-1$, using the following equation:

$$e_{4j+l}(i) = |e_j(i-1) + 2^{i-1} \times E_l|, \tag{3}$$

where $l \in \{0,1,2,3\}$, the parameter j represents the indices of leaves at level $i-1$, and $0 \leq j \leq 2^{2(i-1)}-1$. Employing Eq. (3), we can restate Eq. (2) as the following:

$$\begin{aligned} MED(n) = \\ \frac{1}{2^{2n}} \times (|e_0(n-1) + 2^{n-1} \times E_0| + |e_0(n-1) + 2^{n-1} \times E_1| + \\ |e_0(n-1) + 2^{n-1} \times E_2| + |e_0(n-1) + 2^{n-1} \times E_3| + ... + \\ |e_{2^{2(n-1)}-1}(n-1) + 2^{n-1} \times E_0| + |e_{2^{2(n-1)}-1}(n-1) + 2^{n-1} \times E_1| \\ + |e_{2^{2(n-1)}-1}(n-1) + 2^{n-1} \times E_2| + \\ |e_{2^{2(n-1)}-1}(n-1) + 2^{n-1} \times E_3|), \end{aligned} \tag{4}$$

which still requires 2^{2n} addition operations. In the following, we first present and prove LEMMA 1, and then, we use it to decrease the number of addition operations in Eq. (4).

Lemma 1. *In an n-bit disjoint approximate adder, comprising identical disjoint single-bit building blocks, if the EDs of the employed single-bit building blocks are all positive, the absolute value operator in Eq. (3) is removed, and it is restated as follows:*

$$e_{4(j-1)+l}(i) = e_j(i-1) + 2^{i-1} \times E_l.$$

Proof. As mentioned in Section 2.1.2, a disjoint single-bit building block possesses four EDs referred to as E_0, E_1, E_2, and E_3, which are equivalent to $e_0(1)$, $e_1(1)$, $e_2(1)$, and $e_3(1)$, respectively. When EDs of the building block are all positive, for $i = 2$ (i.e., at level 2), both the terms $e_j(i-1)$ and E_l, where $j,l \in \{0,1,2,3\}$, are positive as well. In this case, the terms within the absolute value operator in Eq. (3) are positive, and thus, the absolute value operator can be removed.

Lets assume that $e_j(i-1)$, which represents the content of each leaf at level $i-1$, is always positive. In this case, since E_0 through E_3 (i.e., E_l) are also positive, the 2^{2i} leaves at level i which are represented by $e_{4j+l}(i)$ in Eq. (3), are positive, regardless of the absolute value operator. Thus, the absolute value operator can be removed. Consequently, by the principle of *Mathematical Induction* [24], the absolute value operator in Eq. (3) can be removed for all i ($2 \leq i \leq n$). □

Based on LEMMA 1, when E_0 through E_3 are all positive, we can remove the absolute value operators in Eq. (4) which leads to the following equation:

$$\begin{aligned} MED(n) &= \frac{1}{2^{2n}} \times [4 \times (e_0(n-1) + ... + e_{2^{2(n-1)}-1}(n-1))] \\ &+ \frac{1}{2^{2n}} \times [2^{2(n-1)} \times 2^{n-1} \times (E_0 + E_1 + E_2 + E_3)] \\ &= MED(n-1) + 2^{n-3} \times (E_0 + E_1 + E_2 + E_3). \end{aligned} \tag{5}$$

Eq. (5) computes MED recursively, using n addition operations, which makes it a lot more efficient compared to Eq. (4). Besides, owing to the fact that Eq. (5) is a *first-order linear* recurrence relation [24], it can be solved using approaches such as the *iteration method*. Solving Eq. (5) results in Eq. (6), which is of the order of $O(1)$.

$$MED(n) = \frac{\sum_{l=0}^{3} E_l}{4} + (2^{n-2} - \frac{1}{2}) \times \sum_{l=0}^{3} E_l \tag{6}$$

Since we have used LEMMA 1 to obtain Eq. (6), this equation can only be used when EDs of building blocks are positive. However, we can follow a similar approach to demonstrate that Eq. (6) can also be used in situations where all EDs are negative, with the only difference that the resulting MED values will also be negative. As a result, we can state that when the EDs of the employed single-bit building blocks are all of the same sign (i.e., all positive or all negative), MED can be computed using Eq. (7).

$$MED(n) = |\frac{\sum_{l=0}^{3} E_l}{4} + (2^{n-2} - \frac{1}{2}) \times \sum_{l=0}^{3} E_l| \tag{7}$$

2.2.2 Second Observation: Importance of Input Patterns

When it comes to selecting an optimal or near optimal adder design for a specific problem (e.g., a DSP application), the design performing the best trade-off within the range of application dictated constraints is always desired. In approximate computing, the compromise is between error criterion (e.g., MED) and circuit characteristics (i.e., power and energy consumption, area occupation, and circuit delay). Error characteristics should be calculated

TABLE 1
Comparison of circuit and error characteristics between LOA and the proposed adder

Design	Dynamic Power (μW)	Static Power (μW)	Area (μm^2)	Delay (fs)	MED (Uniform)
LOA	2.76	0.226	10.108	0.06	47.875
Proposed	2.63	0.195	7.714	0.09	49.32

in a way that describes the error behavior of the selected designs, based on applications input data set. In other words, input patterns should be taken into consideration while selecting a design for a specific application. In the following, we first describe MED computation based a specified input pattern and then present an example to demonstrate the impact of input patterns on the problem of finding optimal adders for a given application

Consider $P(x)$ as the probability of x to be 1, where x can be any single bit of the two binary numbers $a = (a_{n-1}a_{n-2}...a_0)$ and $b = (b_{n-1}b_{n-2}...b_0)$, which are the inputs to an n-bit adder. Consequently, $\overline{P(x)} = 1 - P(x)$ is the probability of x to be 0. Based on this notation, every input pattern can be demonstrated by two sequences $Pat(a) = (P(a_{n-1}), P(a_{n-2}), ..., P(a_0))$ and $Pat(b) = (P(b_{n-1}), P(b_{n-2}), ..., P(b_0))$. All together, these two sequences indicate the probability of each single bit of the two inputs a and b to be 1. For instance, the two sequences $Pat(a) = (0.1, 0.3, 0.6)$ and $Pat(b) = (0.7, 0.4, 0.9)$ demonstrate the pattern of two inputs to a 3-bit adder, according to which $P(a_0) = 0.6$. To compute MED for a given input pattern, we can use Eq. (1), where the probability of each input pair (i.e., Q_j) is computed by Eq. (8).

$$Q_j = \prod_{q=0}^{n-1} F(a_q) \times F(b_q)$$

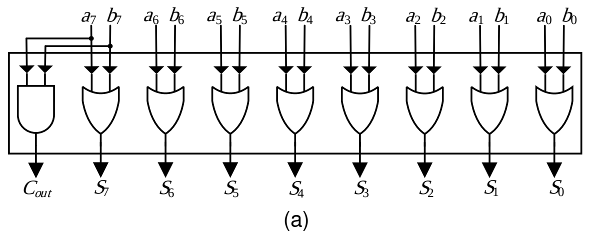
$$F(x) = \begin{cases} P(x) & x = 1 \\ \overline{P(x)} & x = 0 \end{cases} \tag{8}$$

Figure 3 shows the two 8-bit energy and area-efficient approximate adders to clarify the importance of input patterns. We consider *Lena* and *F16* images as inputs to the image addition benchmark. Table 1 shows the MED of the two adders for uniformly distributed inputs and their power, area, and delay by using Verilog description and synthesis in Synopsys Design Compiler with a NanGate FreePDK45nm library [25].

If we consider the MED as our criterion, LOA is selected as the best choice for the image addition. However, the results in Table 2 and Figure 4, demonstrate the superior performance of the proposed adder in both mathematical and subjective metrics, respectively. Although the proposed adder has higher MED than LOA for uniformly distributed inputs, it has better MED for the applied input images. The MED of LOA is 58.99 for the input patterns of Lena and F16, but the MED of the proposed adder is 35.22 for the same input images.

2.2.3 Third Observation: MED Calculation for different Input Patterns

In this subsection, we demonstrate how to obtain MED of approximate adders for a given pair input patterns by utilizing the quad-tree representation.


a_7 b_7 a_6 b_6 a_5 b_5 a_4 b_4 a_3 b_3 a_2 b_2 a_1 b_1 a_0 b_0
C_{out} S_7 S_6 S_5 S_4 S_3 S_2 S_1 S_0
(a)

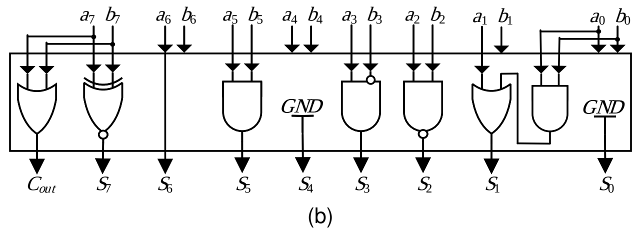

a_7 b_7 a_6 b_6 a_5 b_5 a_4 b_4 a_3 b_3 a_2 b_2 a_1 b_1 a_0 b_0
GND
GND
C_{out} S_7 S_6 S_5 S_4 S_3 S_2 S_1 S_0
(b)

Fig. 3. Gate-level implementation of 8-bit LOA and the proposed 8-bit approximate adder. (a) LOA; (b) the proposed adder.

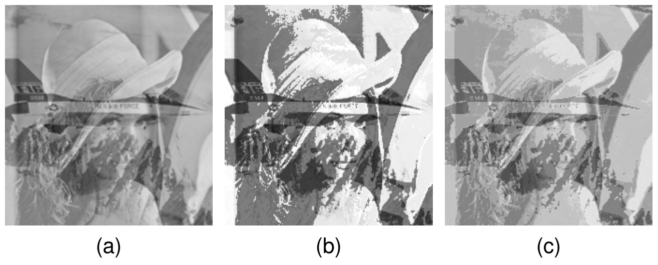


Fig. 4. The results of performing the image addition operation. (a) Accurate 8-bit adder; (b) 8-bit LOA; (c) proposed 8-bit adder in Figure 3b.

Based on the quad-tree representation in Figure 2, leaf j at the last level (i.e, level n) contains the ED of the j^{th} input pair to the adder, where $j = (a_0b_0a_1b_1...a_{n-2}b_{n-2}a_{n-1}b_{n-1})_{10}$. Considering this fact and using Eq. (8), for a given input pattern provided by two sequences $Pat(a) = (P(a_{n-1}), P(a_{n-2}), ..., P(a_0))$ and $Pat(b) = (P(b_{n-1}), P(b_{n-2}), ..., P(b_0))$, MED can be computed using Eq. (9).

$$\begin{aligned} MED(n) = & \\ & [\overline{P(a_0)}\,\overline{P(b_0)}...\,\overline{P(a_{n-1})}\,\overline{P(b_{n-1})}] \times e_0(n) + \\ & ...+ \\ & [P(a_0)P(b_0)...P(a_{n-1})P(b_{n-1})] \times e_{2^{2n}-1}(n) \end{aligned} \tag{9}$$

TABLE 2
Comparison of output quality and MED of LOA and the proposed adder in an image addition application

Design	PSNR	MSE	MAE	MED (Lena and F16)
LOA	9.705	6958	71.39	58.99
Proposed	15.291	1922	34.25	35.22

Based on Eq. (3) and LEMMA 1, we can restate Eq. (9) as:

$$
\begin{aligned}
MED(n) = {} & [\overline{P(a_0)}\,\overline{P(b_0)}...\overline{P(a_{n-2})}\,\overline{P(b_{n-2})}] \times [e_0(n-1) \\
& \times (\overline{P(a_{n-1})}\,\overline{P(b_{n-1})} + \overline{P(a_n)}P(b_n) + P(a_{n-1})\overline{P(b_{n-1})} \\
& + P(a_{n-1})P(b_{n-1})) + 2^{n-1} \times (\overline{P(a_{n-1})}\,\overline{P(b_{n-1})}E_0 \\
& + \overline{P(a_{n-1})}P(b_{n-1})E_1 + P(a_{n-1})\overline{P(b_{n-1})}E_2 \\
& + P(a_{n-1})P(b_{n-1})E_3)] + ... \\
& + [\overline{P(a_0)}\,\overline{P(b_0)}...\overline{P(a_{n-2})}\,\overline{P(b_{n-2})}] \times [e_{2^{2(n-1)}-1}(n-1)\times \\
& (\overline{P(a_{n-1})}\,\overline{P(b_{n-1})} + \overline{P(a_{n-1})}P(b_{n-1}) + P(a_{n-1})\overline{P(b_{n-1})} \\
& + P(a_{n-1})P(b_{n-1}))2^{n-1} \times (\overline{P(a_{n-1})}\,\overline{P(b_{n-1})}E_0 \\
& + \overline{P(a_{n-1})}P(b_{n-1})E_1 + P(a_n)\overline{P(b_{n-1})}E_2 \\
& + P(a_{n-1})P(b_{n-1})E_3)],
\end{aligned}
$$

and since $\forall i \in [0, n-1] : (\overline{P(a_i)}\,\overline{P(b_i)} + \overline{P(a_i)}P(b_i) + P(a_i)\overline{P(b_i)} + P(a_i)P(b_i) = 1)$, it is simplified to the following form:

$$
\begin{aligned}
MED(n) = {} & [\overline{P(a_0)}\,\overline{P(b_0)}...\overline{P(a_{n-2})}\,\overline{P(b_{n-2})}e_0(n-2)] + \\
& [\overline{P(a_0)}\,\overline{P(b_0)}...\overline{P(a_{n-2})}\,\overline{P(b_{n-2})} \times 2^{n-1} \times \\
& (\overline{P(a_{n-1})}\,\overline{P(b_{n-1})}E_0 + \overline{P(a_{n-1})}P(b_{n-1})E_1 + \\
& P(a_{n-1})\overline{P(b_{n-1})}E_2 + P(a_{n-1})P(b_{n-1})E_3)] + ... + \\
& [P(a_0)P(b_0)...P(a_{n-2})P(b_{n-2})e_{2^{2(n-1)}-1}(n-1)] + \\
& [P(a_0)P(b_0)...P(a_{n-2})P(b_{n-2}) \times 2^{n-1} \times (\overline{P(a_{n-1})} \\
& \overline{P(b_{n-1})}E_0 + \overline{P(a_{n-1})}P(b_{n-1})E_1 + P(a_{n-1})\overline{P(b_{n-1})}E_2 + \\
& P(a_{n-1})P(b_{n-1})E_3)]. \qquad (10)
\end{aligned}
$$

Considering the characteristics of the quad-tree representation shown in Figure 2, level $n-1$ represents a disjoint approximate adder with the length of $n-1$, which takes $a = (a_{n-2}...a_1a_0)$ and $b = (b_{n-2}...b_1b_0)$ as inputs. Thus, similar to Eq. (9), the MED of the adder at level $n-1$ can be computed by the following equation:

$$
\begin{aligned}
MED(n-1) = {} & [\overline{P(a_0)}\,\overline{P(b_0)}...\overline{P(a_{n-2})}\,\overline{P(b_{n-2})}e_0(n-1)] \\
& + ... + \\
& [P(a_0)P(b_0)...P(a_{n-2})P(b_{n-2})e_{2^{2(n-1)}-1}(n-1)] \qquad (11)
\end{aligned}
$$

On the other hand, for the adder at level $n-1$ we have:

$$
\begin{aligned}
& \overline{P(a_0)}\,\overline{P(b_0)}...\overline{P(a_{n-2})}\,\overline{P(b_{n-2})} + \\
& \overline{P(a_0)}\,\overline{P(b_0)}...\overline{P(a_{n-2})}P(b_{n-2}) + ... + \\
& P(a_0)P(b_0)...P(a_{n-2})\overline{P(b_{n-2})} + \\
& P(a_0)P(b_0)...P(a_{n-2})P(b_{n-2}) = 1 \qquad (12)
\end{aligned}
$$

Based on Eq. (11) and Eq. (12), we can simplify Eq. (10) to the following form:

$$
\begin{aligned}
MED(n) = {} & MED(n-1) + 2^{n-1} \times [\overline{P(a_{n-1})}\,\overline{P(b_{n-1})}E_0 + \\
& \overline{P(a_{n-1})}P(b_{n-1})E_1 + P(a_{n-1})\overline{P(b_{n-1})}E_2 + \\
& P(a_{n-1})P(b_{n-1})E_3] \qquad (13)
\end{aligned}
$$

For a given input pattern, Eq. (13), which is of the order of $O(n)$, recursively computes MED for the family of adders illustrated in Figure 2. Since we have used LEMMA 1 to obtain Eq. (13), it only works when EDs of building blocks are positive. However, it can be shown that Eq. (13) can also be used when EDs of building blocks are all negative. In this case, the resulting MED is also negative, and thus, its absolute value represents MED.

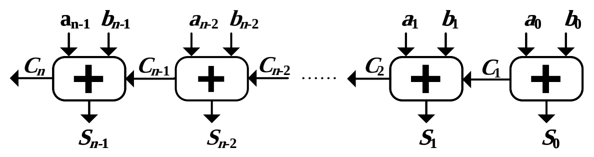


Fig. 5. General structure of LPAAs.

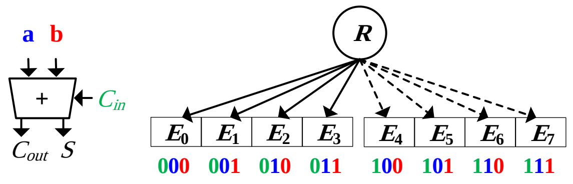


Fig. 6. Extended quad-tree representation of single-bit building blocks with carry signals.

As we can see, the quad-tree representation has the potential to be used as a framework to compute error-metrics such as MED, for both uniform and specified input patterns. However, it has two limitations. First, when the EDs of single-bit building blocks have different signs, MED computation based on the quad-tree representation is not very straightforward, and we had to use LEMMA 1 to be able to compute MED for specific cases. Second, the quad-tree representation can only model disjoint approximate adders. Thus, it can not be used to find formulae for error-metrics while adders comprise building blocks with carry signals among them. Therefore, in Section 3, we address both of these problems by extending the quad-tree representation.

3 PROPOSED APPROACH

In this section, we first show how to use the quad-tree to model carry propagation and EDs with different signs. Next, we demonstrate an accurate calculation of the MED and EP of approximate adders for any given input pattern. At last, a case study is provided to illustrate an up-close and detailed example of the proposed approach. Figure 5 shows the most general structure of an LPAA which we consider as our baseline. In Figure 5, except for the first approximate full adder in the first bit position which does not have a carry-in input, full adders in all other bit positions may have one. Thus, all blocks may generate carry-out signals. Moreover, each of them can have a different configuration compared to the other ones. Also, Due to their different implementation and functionality, there may be EDs with different signs. Considering all possible configurations, including those with different sign EDs, there are 4^8 separate approximate single-bit full adders available. Thus, for an 8-bit LPAA, the design space contains 65536^8 different configurations but Eq. (7) can calculate MED for only a small fraction of this rich design space.

Modeling carry propagation: To model adders with carry-out signals, we extend the quad-tree of Figure 1, as Figure 6. As shown in Figure 6, approximate full adders receive three inputs, so they have eight edges. Thus, they should be modeled with eight different EDs (the dashed-lines refer to the case where carry-in = 1). Depending on the carry-in input, only four of those edges are used at a time. Figure 7 shows the quad-tree of two consecutive single-bit full adders forming a 2-bit LPAA. Table 3 shows the

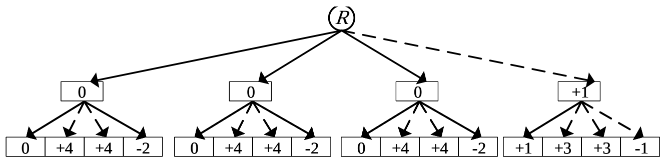


Fig. 7. An example of a 2-bit non-disjoint LPAA uisng modified quad-tree representation.

TABLE 3
Approximate adder at bit position 1.

♯	a	b	s	C_{out}	ED
1	0	0	0	0	0
2	0	1	1	0	0
3	1	0	1	0	0
4	1	1	1	1	+1

truth table of the single-bit full adder at bit position 1 and Table 4 shows the truth table of the single-bit full adder at bit position 2.

Table 3 generates a carry-out signal only for the case it receives 11 as its inputs, otherwise, its carry-out is zero. So, if both a_0 and b_0 are 1, then the four selected edges showing the functionality of the second block, are those having carry-in of 1 (i.e., the four bottom rows in Table 4). But, if a_0 and b_0 are not 11, then the other four edges, the top four rows of Table 4, are selected to represent the functionality of this block.

Modeling EDs with different signs: Consider the example 2-bit LPAA in Figure 7. Four of the leaves are negative. Therefore, there are EDs with different signs and LEMMA 1 is no longer applicable. Hence, we cannot use Eq. (7) for MED computation. To overcome this problem, we aim on dividing MED into several groups.

For example, the MED of the 2-bit LPAA can be divided into MED of positive and negative nodes, and each of them can be calculated, separately. Finally, the total MED is the sum of both MED of positive and negative nodes. In a more generic manner, to classify MED into several smaller groups, we need to find out all interactions between two consecutive approximate full adders at bit positions $i-1$ and i (i.e., parent nodes and their connecting edges to their child nodes).

3.1 Node grouping

Based on the quad-tree representation, at the depth $i-1$, there are $2^{2(i-1)}$ nodes. We classify these nodes into 14 separate groups, depending on carry-out signals, the sign of their EDs, and their interaction with their edges.

Therefore, nodes, based on their carry-out signal, can be classified into two groups (i.e., carry-out = 1, and carry-out = 0). Depending on their sign, they can be classified into three groups (i.e., positive, negative, and zero). Also, for the positive and negative cases, they may have the absolute ED value larger, equal, or less than their edges with different sign. Therefore, again, we classify them into three other groups (namely, *Normal*, *Abnormal0*, *Abnormal1*):

1) *Normal*: parents at depth $i-1$, where their absolute EDs are less than 2^{i-1}. In this case, the sign of the child node is always determined by the sign of the edge that connects it to its parent.

TABLE 4
Approximate adder at bit position 2

♯	C_{in}	a	b	s	C_{out}	ED
1	0	0	0	0	0	0
2	0	0	1	1	1	+2
3	0	1	0	1	1	+2
4	0	1	1	1	0	-1
5	1	0	0	1	0	0
6	1	0	1	1	1	+1
7	1	1	0	1	1	+1
8	1	1	1	0	1	-1

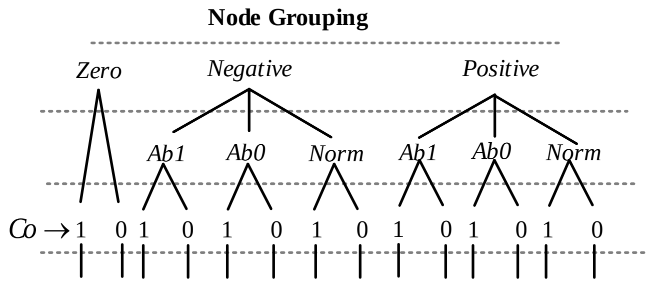


Fig. 8. Node grouping: all 14 groups of nodes.

2) *Abnormal0*: refers to the parents at depth i-1, with their absolute EDs exactly equal to 2^{i-1}. In this case, based on the amount of the parents edge, the child node is zero.
3) *Abnormal1*: refers to the parents at depth i-1, with their absolute EDs greater than 2^{i-1}. In this case the child nodes sign can be the same as its connecting edges sign, or its parents sign.

In other words, for a single-bit approximate full adder, nodes with the ED value of ±1, are normal. ED value of ±2 refers to abnormal0 nodes and abnormal1 nodes are identified with ED value of ±3.

Since the absolute amount of leaves determine MED, we use this classification and describe MED in a more fine-grained manner by dividing MED into 12 portions (two zero groups are neglected for MED calculation) in a way that the sum of all parts is equal to MED. Table 5, shows the classified MED of an n-bit LPAA and its portions (Indexes "P", "N", and "Z" stand for positive, negative, and zero, respectively. Normal, abnormal0, and abnormal1 are shown with "$Norm$", "$Ab0$", and "$Ab1$").

3.2 Edge grouping

This process is quite the same as node classification except that the carry-in signal should be taken into considerations too. So, based on receiving carry-in and generating carry-out, there are four groups. Again, we classify the edges based on their sign to be positive, negative, or zero, into three groups. Finally, based on the value of the edge that connects the current child node to its parent, there are three more groups formed, called Normal, Abnormal0, and Abnormal1. Thus, there are 28 different types of edges available, shown in Table 6 (to save space, we only show normal edges in the table). Since these edges describe the behavior of an approximate adder in the current bit position (i.e., at depth

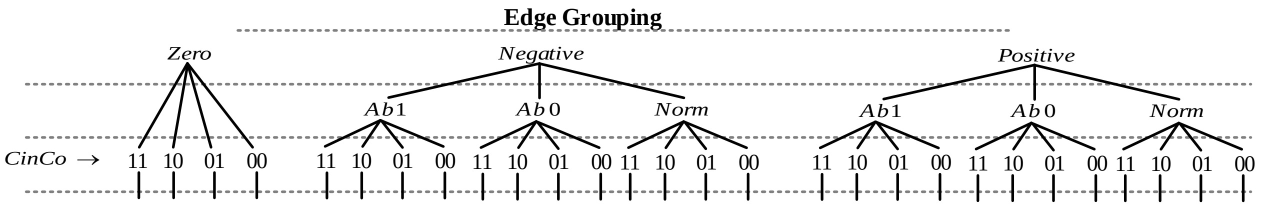


Fig. 9. Edge grouping: all 28 groups of edges.

TABLE 5
Dividing MED into 12 portions

Portion name	Description
$MED_P(Norm, \overline{Co}, n)$	MED of Positive Normal leaves with Carry-out = 0
$MED_P(Norm, Co, n)$	MED of Positive Normal leaves with Carry-out = 1
$MED_P(Ab0, \overline{Co}, n)$	MED of Positive Abnormal0 leaves with Carry-out = 0
$MED_P(Ab0, Co, n)$	MED of Positive Abnormal0 leaves with Carry-out = 1
$MED_P(Ab1, \overline{Co}, n)$	MED of Positive Abnormal1 leaves with Carry-out = 0
$MED_P(Ab1, Co, n)$	MED of Positive Abnormal1 leaves with Carry-out = 1
$MED_N(Norm, \overline{Co}, n)$	MED of Negative Normal leaves with Carry-out = 0
$MED_N(Norm, Co, n)$	MED of Negative Normal leaves with Carry-out = 1
$MED_N(Ab0, \overline{Co}, n)$	MED of Negative Abnormal0 leaves with Carry-out = 0
$MED_N(Ab0, Co, n)$	MED of Negative Abnormal0 leaves with Carry-out = 1
$MED_N(Ab1, \overline{Co}, n)$	MED of Negative Abnormal1 leaves with Carry-out = 0
$MED_N(Ab1, Co, n)$	MED of Negative Abnormal1 leaves with Carry-out = 1

i), the edge variables starts "CR". There are two different kinds of edge variables; (1) CR variables refer to the weighted sum of specific edge errors; and (2) $CRNUM$ variables are equal to the occurrence probability of a specific edge error. Figure 10 shows how to calculate all 12 portions of positive CR variables for a given approximate full adder and a given input pattern. The algorithm in Figure 10 takes the truth table and the probabilities of its input pair as its inputs and outputs all 12 portions of "CR_P". It observes all the eight possible input combinations and their errors. Then, based on their input pattern probability, the amount of errors, carry-in, and carry-out signals, the amount of each of the classes of "CR_P" is calculated. Since the for loop in line 7, iterates 8 times at the most, then all "CR_P" variables are calculated with $O(8)$. The same process should be accomplished for CR_N, too. But for zero nodes CR_Z is always equal to 0.

Having 14 groups for parent nodes and 28 groups for edges, we end up having 392 different types of interactions between a node and its connecting edge. However, there are a lot of invalid interactions among these 392 types. There are three categories for invalid interactions: (1) invalid parents, (2) invalid edges, (3) invalid connections. These invalid interactions and their conditions are as followings:

```
Input: $TruthTable_{Approximate}$, $P(a)$, $P(b)$
Output: All 12 portions of $CR_P$
1.  $Err = TruthTable_{Approximate} - TruthTable_{Accurate}$
2.  if This adder recieves carry-in signal then
3.      $Combinations \leftarrow 8$
4.  else
5.      $Combinations \leftarrow 4$
6.  end if
7.  for $i = 1$ to $Combinations$ do
8.      if $a == 0$ and $b == 0$ then
9.          $Probability \leftarrow \overline{P(a)} \times \overline{P(b)}$
10.     else if $a == 0$ and $b == 1$ then
11.         $Probability \leftarrow \overline{P(a)} \times P(b)$
12.     else if $a == 1$ and $b == 0$ then
13.         $Probability \leftarrow P(a) \times \overline{P(b)}$
14.     else if $a == 1$ and $b == 1$ then
15.         $Probability \leftarrow P(a) \times P(b)$
16.     end if
17.     if $Err(i) == +1$ then
18.         if $\overline{Cin}$ and $\overline{Co}$ then
19.             $CR_P(Norm, \overline{Cin}, \overline{Co}) += Probability \times |Err(i)|$
20.         else if $\overline{Cin}$ and $Co$ then
21.             $CR_P(Norm, \overline{C_{in}}, Co) += Probability \times |Err(i)|$
22.         else if $Cin$ and $\overline{Co}$ then
23.             $CR_P(Norm, Cin, \overline{Co}) += Probability \times |Err(i)|$
24.         else if $Cin$ and $Co$ then
25.             $CR_P(Norm, Cin, Co) += Probability \times |Err(i)|$
26.         end if
27.     else if $Err(i) == +2$ then
28.         if $\overline{Cin}$ and $\overline{Co}$ then
29.             $CR_P(Ab0, \overline{Cin}, \overline{Co}) += Probability \times |Err(i)|$
30.         else if $\overline{Cin}$ and $Co$ then
31.             $CR_P(Ab0, \overline{C_{in}}, Co) += Probability \times |Err(i)|$
32.         else if $Cin$ and $\overline{Co}$ then
33.             $CR_P(Ab0, Cin, \overline{Co}) += Probability \times |Err(i)|$
34.         else if $Cin$ and $Co$ then
35.             $CR_P(Ab0, Cin, Co) += Probability \times |Err(i)|$
36.         end if
37.     else if $Err(i) == +3$ then
38.         if $\overline{Cin}$ and $\overline{Co}$ then
39.             $CR_P(Ab1, \overline{Cin}, \overline{Co}) += Probability \times |Err(i)|$
40.         else if $\overline{Cin}$ and $Co$ then
41.             $CR_P(Ab1, \overline{C_{in}}, Co) += Probability \times |Err(i)|$
42.         else if $Cin$ and $\overline{Co}$ then
43.             $CR_P(Ab1, Cin, \overline{Co}) += Probability \times |Err(i)|$
44.         else if $Cin$ and $Co$ then
45.             $CR_P(Ab1, Cin, Co) += Probability \times |Err(i)|$
46.         end if
47.     end if
48. end for
```

Fig. 10. Calculation of all 12 portions of CR_P for a given approximate full adder

1. Invalid parents: Positive abnormal0 and abnormal1 parents

TABLE 6
Classification of normal edges

Portion name	Description
$CR_P(Norm, \overline{C_{in}}, \overline{Co})$	Weighted Sum of Positive Normal edges with Carry-in=0 & Carry-Out=0
$CR_P(Norm, \overline{C_{in}}, Co)$	Weighted Sum of Positive Normal edges with Carry-in=0 & Carry-Out=1
$CR_P(Norm, C_{in}, \overline{Co})$	Weighted Sum of Positive Normal edges with Carry-in=1 & Carry-Out=0
$CR_P(Norm, C_{in}, Co)$	Weighted Sum of Positive Normal edges with Carry-in=1 & Carry-Out=1
$CR_N(Norm, \overline{C_{in}}, \overline{Co})$	Weighted Sum of Negative Normal edges with Carry-in=0 & Carry-Out=0
$CR_N(Norm, \overline{C_{in}}, Co)$	Weighted Sum of Negative Normal edges with Carry-in=0 & Carry-Out=1
$CR_N(Norm, C_{in}, \overline{Co})$	Weighted Sum of Negative Normal edges with Carry-in=1 & Carry-Out=0
$CR_N(Norm, C_{in}, Co)$	Weighted Sum of Negative Normal edges with Carry-in=1 & Carry-Out=1

TABLE 7
Normal positive and zero parents interacting with different classes of positive edges

parent	edge	child
$Pos(Norm, \overline{Co}, i-1)$	$Pos(Norm, \overline{Cin}, \overline{Co})$	$Pos(Norm, \overline{Co}, i)$
	$Pos(Ab0, \overline{Cin}, \overline{Co})$	Invalid Edge
	$Pos(Ab1, \overline{Cin}, \overline{Co})$	Invalid Edge
	$Pos(Norm, \overline{Cin}, Co)$	$Pos(Norm, Co, i)$
	$Pos(Ab0, \overline{Cin}, Co)$	$Pos(Ab1, Co, i)$
	$Pos(Ab1, \overline{Cin}, Co)$	$Pos(Ab1, Co, i)$
$Pos(Norm, Co, i-1)$	$Pos(Norm, \overline{Cin}, \overline{Co})$	Invalid Edge
	$Pos(Ab0, \overline{Cin}, \overline{Co})$	Invalid Edge
	$Pos(Ab1, \overline{Cin}, \overline{Co})$	Invalid Edge
	$Pos(Norm, \overline{Cin}, Co)$	$Pos(Norm, Co, i)$
	$Pos(Ab0, \overline{Cin}, Co)$	$Pos(Ab1, Co, i)$
	$Pos(Ab1, \overline{Cin}, Co)$	Invalid Edge
$Zero(\overline{Co}, i-1)$	$Pos(Norm, \overline{Cin}, \overline{Co})$	$Pos(Norm, \overline{Co}, i)$
	$Pos(Ab0, \overline{Cin}, \overline{Co})$	Invalid Edge
	$Pos(Ab1, \overline{Cin}, \overline{Co})$	Invalid Edge
	$Pos(Norm, \overline{Cin}, Co)$	$Pos(Norm, Co, i)$
	$Pos(Ab0, \overline{Cin}, Co)$	$Pos(Ab0, Co, i)$
	$Pos(Ab1, \overline{Cin}, Co)$	$Pos(Ab1, Co, i)$
$Zero(Co, i-1)$	$Pos(Norm, \overline{Cin}, \overline{Co})$	Invalid Edge
	$Pos(Ab0, \overline{Cin}, \overline{Co})$	Invalid Edge
	$Pos(Ab1, \overline{Cin}, \overline{Co})$	Invalid Edge
	$Pos(Norm, \overline{Cin}, Co)$	$Pos(Norm, Co, i)$
	$Pos(Ab0, \overline{Cin}, Co)$	$Pos(Ab0, Co, i)$
	$Pos(Ab1, \overline{Cin}, Co)$	Invalid Edge

that do not generate carry-out do not exist. Negative abnormal0 and abnormal1 parents that generate carry-out do not exist either.

2. Invalid edges:

2.1) A positive normal edge with carry-in of 1 and no carry out does not exist. Similarly, a negative normal edge with no carry-in and carry-out of 1, does not exist.

2.2) There are no positive abnormal0 edges that do not generate a carry-out. Also, there are no positive abnormal1 edges that have no carry-out or receives a carry-in of 1.

2.3) A negative abnormal0 edge that generates a carry-out does not exist. A negative abnormal1 edge that generates carry-out or does not receive a carry-out does not exist.

3. Invalid connections: A parent node that has carry-out of 1 cannot interact with an edge that does not receive a carry-in. Also, a parent node that has no carry-out cannot interact with an edge with carry-in of 1.

After examining all possible 14×28 (392) interactions, we find out that 312 of them are invalid. Therefore, only 80 interactions are possible between a parent not and its edges. For each of these 80 interactions between parents and edges, we must identify the group to which the offspring nodes belong. Then, each one of the 12 portions of MED can be calculated separately. As an example, Table 7 shows the child node classes when normal positive or zero parents interact with different classes of positive edges (Note that invalid connections are removed from the table).

3.3 Case Study

As an example, assume that we want to calculate the desired portion for bit-position i ($MED_P(Ab0, Co, i)$). Therefore, we need all MED and EP portions at bit-position i-1. Also, we need all CR and $CRNUM$ variables for the i^{th} (current) block. The idea is to add the errors of the i^{th} block to the cumulative amount (from bit-positions 0 to i-1) of desired portion of the MED for bit-position i. Thus, we must first identify all interactions, at bit position i-1 that are leading to positive, abnormal nodes with carry-out at bit-position i. Table 8 shows identifies all four interactions that leads to child nodes of group $P(Ab0, Co)$. As an example, the first row of Table 8 indicates that for each $P(Ab0, Co)$ nodes at i-1, there is one edge (i.e., $P(Norm, Cin, Co)$) that leads to a $P(Ab0, Co)$ child node at bit-position i.

Before calculating the desired portion of MED at bit position i, we must know how to compute the contributed EDs from parents and their edges to the desired portion. Based on Eq. 3, the error of a child node can be considered as the absolute summation (or subtraction) between the term $e_j(i-1)$ and the term $2^{i-1} \times E_l$. Our approach for computing the error of each interaction is to calculate each of the two terms separately, and then add them together.

In Eq. 3, the left hand side term, $e_{4j+l}(i)$ is the total error of a specific interaction, which is also referred to as a child error. In the quad-tree representation, this error is written inside a child node at level i. The term, $e_j(i-1)$ is the error of the parent which also indicates that the errors of the parents are propagated to the lower levels. In other words, this term contains the error that the parent is contributing.The term, $2^{i-1} \times E_l$, is the multiplication between the error of an edge and its corresponding weight. This is because when calculating the variables related to edges (Figure 10), we neglect their bit-positions. This term contains the error that the edge (or the current block) contributes. After computing each term, if they do not have same signs, the error of the child is equal to the absolute amount subtraction between $e_j(i-1)$ and $2^{i-1} \times E_l$. Otherwise, the error of the child is the absolute summation of the them.

Now for each of the interactions shown in Table 8, we first calculate the parent error. Then, we show how to compute the edge errors, at their bit-position (i.e., the term $2^{i-1} \times E_l$). Form row 1 in Table 8, we understand that trough an interaction between a $P(Ab0, Co, i-1)$ parent and a $P(Norm, Cin, Co)$, a positive abnormal0 child node with a carry-out is formed at level i. Therefore, to calculate the parent errors, we need two

TABLE 8
Interactions that lead to a positive abnormal0 children with carry-out of 1

♯	Parent	Edge
1	$Positive(Ab0, Co, i-1)$	$Positive(Norm, Cin, Co)$
2	$Negative(Ab0, \overline{Co}, i-1)$	$Positive(Ab1, \overline{Cin}, Co)$
3	$Zero(\overline{Co}, i-1)$	$Positive(Ab0, \overline{Cin}, Co)$
4	$Zero(Co, i-1)$	$Positive(Ab0, Cin, Co)$

values; (1) the sum of all $P(Ab0, Co, i-1)$ parents; (2) and, the occurrence probability of a $P(Norm, Cin, Co)$ edges of the current adder. By multiplying both of them, we can compute the error amount that all the $P(Ab0, Co, i-1)$ parents contribute trough their $P(Norm, Cin, Co)$ edges. The first value is equal to $MED_P(Ab0, Co, i-1)$ and the second value is $CRNUM_P(Norm, Cin, Co)$. We assume that we have all previous MED and EP portions, and the CR variables are all computed in constant time and space. Therefore, the parent errors of the first interaction of Table 8 can be calculated as Eq. (14).

$$MED_P(Ab0, Co, i{-}1) \times CRNUM_P(Norm, Cin, Co, i) \tag{14}$$

Similarly, the parent errors for row 2 can be calculated as Eq. 15.

$$MED_N(Ab0, \overline{Co}, i-1) \times CRNUM_P(Ab1, \overline{Cin}, Co, i) \tag{15}$$

For rows 3 and 4, the $MED_Z(\overline{Co}, i{-}1)$ and $MED_Z(Co, i{-}1)$ are both zero. Therefore, for these two rows the parent errors are zero. To compute the errors caused by edges for the first row, two values must be available; (1) the occurrence probability of all $P(Ab0, Co, i-1)$ parents; (2) and, the sum of errors of $P(Norm, Cin, Co)$ edges of the current block. The product of these two is equivalent to calculating the edge errors for all $P(Ab0, Co, i-1)$ parents. The first value is equal to $EP_P(Ab0, Co, i-1)$, and the second value refers to $CR_P(Norm, Cin, Co)$. Thus, both values are available. Therefore, the edge errors for row 1 of Table 8 are calculated as Eq 16.

$$EP_P(Ab0, Co, i-1) \times CR_P(Norm, Cin, Co, i) \times 2^{bitposition} \tag{16}$$

Similarly, Eq 17 shows the edge errors for row 2 of Table 8.

$$EP_N(Ab0, \overline{Co}, i-1) \times CR_P(Ab1, \overline{Cin}, Co, i) \times 2^{bitposition} \tag{17}$$

And the edge errors for rows 3 and 4 are calculated as Eq 18 and Eq 19.

$$EP_Z(\overline{Co}, i-1) \times CR_P(Ab0, \overline{Cin}, Co, i) \times 2^{bitposition} \tag{18}$$

$$EP_Z(Co, i-1) \times CR_P(Ab0, Cin, Co, i) \times 2^{bitposition} \tag{19}$$

So far, the total parent errors and total the edge errors of all four interactions have been calculated using Eq. 14 to Eq. 19. For each interaction, the parent and the edge errors must be added/subtracted with/from each other. If they have the same signs, the total error of that interactions is equal to the absolute amount of their summation. Otherwise, they are subtracted from each other. For the first row, both the parent and the edge are positive. Therefore, the total error that row 1 contributes to $MED_P(Ab0, Co, i)$ is equal to absolute summation of Eq. 14 and Eq 16. For the interaction in the second row of Table 8, the parent and the edge have different signs. Thus, the total error that this interaction is contributing to $MED_P(Ab0, Co, i)$ is equal to the absolute difference of Eq. 15 and Eq. 17. For row 3 and row 4, the parent errors is zero, hence, the contributed errors of these interactions is equal to their corresponding edge errors, Eq. 18 and Eq. 19, respectively. Finally, the $MED_P(Ab0, Co, i)$ is equal to the summation of total error of all four interactions.

In above equations, we assume that we have all MED and EP portions for the previous level i-1, and we demonstrate how to calculate MED at level i. Here we show the calculation of $EP_P(Ab0, Co, i)$ portions for level i. By definition, $EP_P(Ab0, Co, i)$ is the occurrence probability of positive abnormal0 nodes with carry-out at level i. Similar to computation of $MED_P(Ab0, Co, i)$, the first step is to identify all interactions that lead to the desired group of nodes at level i (i.e., positive abnormal0 nodes with carry-out), which is shown in Table 8. Now we start calculating the contribution of each of the four rows/interactions in Table 8 to $EP_P(Ab0, Co, i)$.

For row 1 of Table 8, to compute this portion of EP, we need two values; (1) the occurrence probability of all $P(Ab0, Co)$ parents at level i-1; and, (2) the probability of a $P(Ab0, Co)$ generating a $P(Ab0, Co)$ child. The amount that the first row/interaction is contributing to $EP_P(Ab0, Co, i)$ is equal to the product of these two values. The first value is equal to $EP_P(Ab0, Co, i)$. Based on Table 8, a $P(Ab0, Co)$ parent at level i-1 generates a $P(Ab0, Co)$ node at level i through its $P(Norm, Cin, Co)$ edges. Therefore, the second value is equal to the occurrence probability of $P(Norm, Cin, Co)$ edges of the i^{th} single-bit adder. The amount row 1 contribute to $EP_P(Ab0, Co, i)$ can be calculated as Eq 20.

$$EP_P(Ab0, Co, i-1) \times CRNUM_P(Norm, Cin, Co, i) \tag{20}$$

Similarly, for rows 2, 3, and 4 of Table 8, we need to multiply the probability of the parent to the probability of its specific edge. Eq 21, 22, and 23, shows the amounts rows 2, 3, and 4 contribute to $EP_P(Ab0, Co, i)$, respectively.

$$EP_N(Ab0, \overline{Co}, i{-}1) \times CRNUM_P(Ab1, \overline{Cin}, Co, i) \tag{21}$$

$$EP_Z(\overline{Co}, i-1) \times CRNUM_P(Ab0, \overline{Cin}, Co, i) \tag{22}$$

$$EP_Z(Co, i-1) \times CRNUM_P(Ab0, Cin, Co, i) \tag{23}$$

By applying the same method to all other 13 portions and adding them together, we can calculate EP with $O(n)$. Thus, for MED calculation, at first, we calculate all portions of EP with $O(n)$. Then, we use EP portions to compute MED, again with $O(n)$.

4 Evaluations

In this section we evaluate the proposed framework for generating approximate circuits automatically. Four experiments have been conducted to thoroughly demonstrate the benefits of the proposed framework. The first experiment shows the speed-up of our proposed method for MED and EP calculation over existing Monte Carlo sampling methods. The second experiment demonstrates the benefit of fast error metric calculation in automatic generation tools dealing with a huge design space. Since our framework can explore the design space much faster than the other ones employing the existing Monte Carlo sampling methods.

In the third experiments, LPAAs are automatically generated in an input pattern-aware manner for a given image processing application that achieves better results in circuit and image processing metrics than the state-of-the-art approximate adders. To demonstrate the generality of the results, the fourth experiment evaluates the generated adders of the third experiment over a wider range of input images. Also, the adders are evaluated in an edge detection application. To conduct fair comparisons between the baseline and AxMAP, all programming and simulations are carried out in MATLAB. Circuit metrics are evaluated using Verilog description alongside with the well-known Synopsys Design Compiler synthesis tool using the NanGate FreePDK45nm library [25].

4.1 AxMAP Methodology

To demonstrate the superiority of the proposed formulae over the MC sampling method, we develop AxMAP that automatically generates efficient application-specific and input-aware approximate adders. AxMAP serves as a framework for evaluating the potentials of input-aware error calculation in the quality of result (QoR). The AxMAP tool takes five input parameters. Power, area, and delay are considered as the designers circuit budget constraints. The tolerable MED over a given pair of input patterns is considered as the error constraint. AxMAP explores the design space and generates approximate adders using the random search algorithm. In each iteration, the selected design satisfying the constraints is considered as a valid output.

Figure 11 shows the input parameters and the proposed basic building blocks for AxMAP. The input parameters to AxMAP are as follow:

1) **Adder Length (L)**. The bit-width of adders to be generated by AxMAP.
2) **Delay Constraint**. A natural number referring to the maximum length of the carry chain allowed for the adders generated by AxMAP.
3) **Input Patterns**. Two L-bit sequences indicating the bit probability distributions of the two inputs of the adders generated by AxMAP.
4) **MED and EP constraints**. Two real numbers specifying the maximum amount of error allowed for a specific application or input patterns.
5) **Power and area constraints**. Two real numbers representing the maximum amount of the power consumption and area occupation of the adders generated by AxMAP.

As illustrated in Figure 11, AxMAP employs six basic building blocks and a *Basic Library* that contains all possible (i.e., 65536) single-bit approximate adders. During the first stage, AxMAP performs a *Random Search* for selecting L single-bit samples from the Basic Library to form an L-bit adder object. *Delay Estimation* is performed in the second stage, where the longest carry chain of the adder object is compared with the input delay constraint. If the latency of carry chain is less than or equal to the input delay constraint, the adder object is passed to the next stage. Otherwise, the tool restarts from the first stage, and it attempts to form another L-bit adder object. We consider a user-defined limit for the total number of restarts in AxMAP. The limit is necessary because of two reasons: (1) an exhaustive search amongst all potential designs is extremely time-consuming and (2) exceedingly optimistic bounds for the power, area, delay, and MED requirements of a design may be impractical within the available technology nodes, thereby making the search impossible to converge. In all our experiments, this restart limit is set to 100000 that forces AxMAP to conclude after 100000 iterations.

An *Accurate MED/EP Calculation* is carried out during the third stage. The MED (or EP) of the adder object is calculated on a given pair of input patterns. Then, the calculated MED is compared with the MED constraint input. If the calculated MED is not less than the MED constraint, AxMAP returns the first stage. Otherwise, it proceeds to the fourth stage for *Power and Area Estimation* by extracting the type and number of gates used for the adder object. If the power or area is more than what user demands, AxMAP returns to the first stage for a new design attempt. The fifth stage is employed to *Generate Verilog HDL Code* for the adder object using a gate-level circuit description stored in a file called adder.v. In the sixth stage, the generated Verilog file is passed to the *Design Compiler* for synthesis and a more accurate estimation of circuit characteristics. If the generated adder still satisfies all of the input circuit constraints, it will be considered as a valid output adder. Therefore, AxMAP records the generated Verilog file along with its circuit and error characteristics. If the circuit constraints are not satisfied, the tool starts the design process over from the first stage to form a new L-bit LPAA.

4.2 Experiment 1: Speedup over the existing methods

Often, in automatic approximate circuit (e.g., adders) generation tools, calculating error metrics takes much more time than that of the circuit and electrical parameters (i.e., power, area and the delay). Thus, a fast error computation method makes the tool much more efficient. Moreover, as the size of generated adders grow, the MC method needs more samples to estimate the error behavior which slows down the tool. Besides, the results that are calculated using the MC are just estimates while accurate evaluations are indeed more reliable.

For the first experiment, we generate seven hundred approximate adders with variable configurations and bit-widths, from 8 to 128-bit (one hundred for each bit-width) wide for AxMAP and the baseline (the baseline is similar to AxMAP but it employs the MC method for MED calculations). The MED of each one of the adders is computed with the MC (with 10000 samples) and the proposed formula over uniform and a randomly generated pair of input patterns. Table 9 shows that AxMAP, while being entirely accurate, is able to calculate the MED over 150x faster than the baseline.

To further clarify the performance of the proposed approach, Figure 12 present its computational cost scalability. We generate more than 150 approximate adders with different bit-width (ten different adders for each bit-width). Then, we calculate their MED and report their corresponding average runtime. Figure 12a shows the average runtime of MED calculation of one adder in the range 4 to 12 bits, with an increment step of 2. As shown in the figure, the runtime of the proposed approach grows linearly with respect to the size of the adder, which supports the claim about the linear

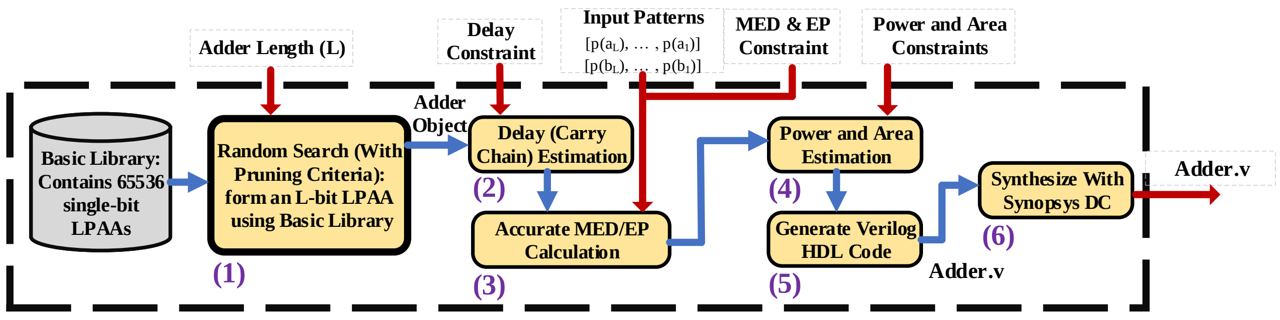

Adder Length (L)
Delay Constraint
Input Patterns [p(a_L), … , p(a_1)] [p(b_L), … , p(b_1)]
MED & EP Constraint
Power and Area Constraints
Basic Library: Contains 65536 single-bit LPAAs
Random Search (With Pruning Criteria): form an L-bit LPAA using Basic Library
(1)
Adder Object
Delay (Carry Chain) Estimation
(2)
Accurate MED/EP Calculation
(3)
Power and Area Estimation
(4)
Generate Verilog HDL Code
(5)
Adder.v
Synthesize With Synopsys DC
(6)
Adder.v

Fig. 11. Block diagram flow of the implemented framework, AxMAP.

TABLE 9
Comparing the speed-up runtime execution of AxMAP over the baseline for 8bit to 128bit wide LPAAs

Length	8	16	24	32	48	64	128
AxMAP (s)	4.09	9.06	10.79	15.39.	22.99	30.35	62.12
Baseline (min)	13.45	24.7	32.3	45	65	77	170
Speed-up	197	163	183	175	169	151	164

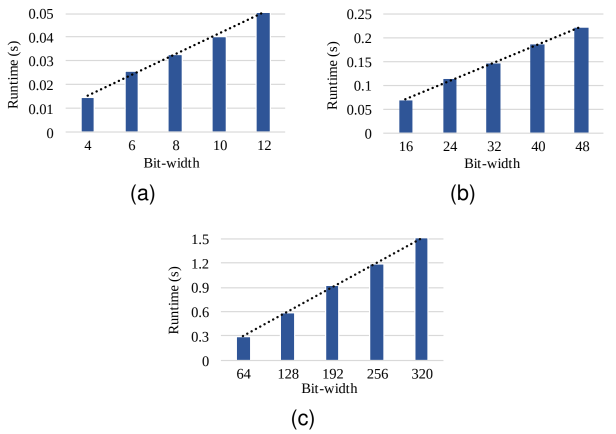


Fig. 12. Cost-scalability analysis. (a) 4 to 12 bits; (b) 16 to 48 bits; (c) 64 to 320 bits.

time complexity of MED calculation. As shown in Figure 12a, a 2-bit growth in the length of adders adds almost 0.01 seconds to the MED computation runtime. Figure 12b and Figure 12c demonstrate the MED runtime of adders in range 16 to 48 and 64 to 320, with steps of 16 and 64, respectively. Based on Figures 12b and 12c, a 16-bit and a 64-bit growth adds 0.05 and 0.3 seconds to the runtime, respectively.

4.3 Experiment 2: More efficient design space exploration

As mentioned in Section 3, the design space of approximate adders is enormous which makes automatic design space exploration tools more crucial than ever. Moreover, an automatic circuit generation tool equipped with a fast quantitative error analysis method can find the desired circuit much faster than the regular ones relying on the MC method-based error analysis. Thus, to illustrate the efficiency that the proposed method grants the automatic adder generation tool (AxMAP), this experiments focuses on exploring the design space of approximate adders to find suitable and efficient approximate adders regarding MED-Power and MED-Area trade-off.

In [26] which is the most complete study in terms of comparing and evaluating the well-known approximate adders, LOA is considered to be one of the best designs, especially when it comes to hardware efficiency (i.e., power consumption and die area occupation). In [27] by A. Najafi et al. which is another comprehensive work in terms of comparing various approximate adders from a refreshing view point, again, LOA is chosen as the best design. Therefore, in our experiments, we choose a set of circuit (power and area) and error characteristics slightly lower than LOA as the constraints of AxMAP. In this case, the outputs of AxMAP surely outperform LOA at least in two or three of the metrics. During the configuration process of AxMAP, we consider the delay constraint equal to delay of a single-bit full adder. Therefore, in the delay-related worst case scenario, the output design delay is at least slightly less than the delay of a single-bit full adder. In less than 12 hours, in a Corei5 2.5GHz processor and 8 GB of RAM, for uniform input patterns, AxMAP was able to generate more than 150 8-bit approximate adders that perform better than LOA in terms of MED, area, and power consumption, only using random searching, which clearly shows the benefit of the proposed fast error analysis method alongside the fact that the design space is filled with useful approximate adders that has never been found.

Figure 13 demonstrates the MED and power-area product diagram of all generated adders of the second experiment. Both MED and power-area product are normalized based on those of LOA. Compared to LOA, all adders demonstrate better performance in terms of MED and power-area product. Three Pareto optimal structures, with IDs #1, #12, and #146, are chosen (distinguished with a red dashed oval in Figure 13) as the best designs. Figure shows the gate-level implementations of these designs (since this experiment was performed over uniform inputs the chosen designs names start with U_ followed by their IDs). Out of three chosen designs, U_1 has the best accuracy with 33% better MED than LOA; and U_146 is the most power-area efficient demonstrating 65% improvements when compared with LOA. U_12 is a trade-off point between U_1 and U_146; and it shows 30% better MED, and 39% less power-area, compared to LOA.

4.4 Experiment 3: Input pattern-aware automatic generation of LPAAs for image addition

Usually, the benefits of employing approximate units are investigated on a real-life benchmark, e.g., an image processing application, such as many of the previous well-known papers including

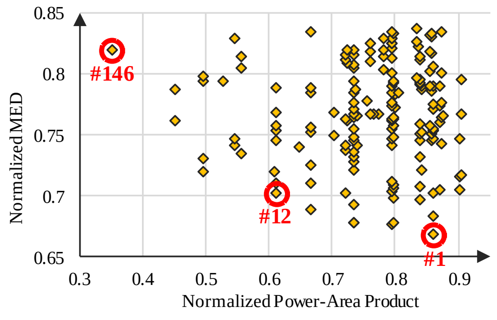


Fig. 13. MED Power-Area Product of generated adders normalized based on those of LOA for uniform input patterns.

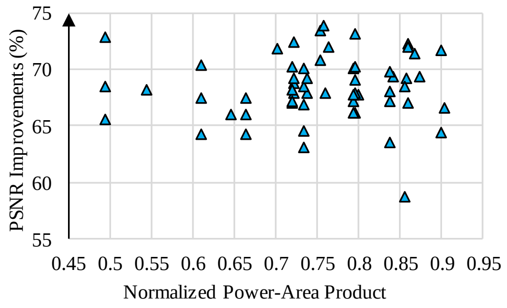
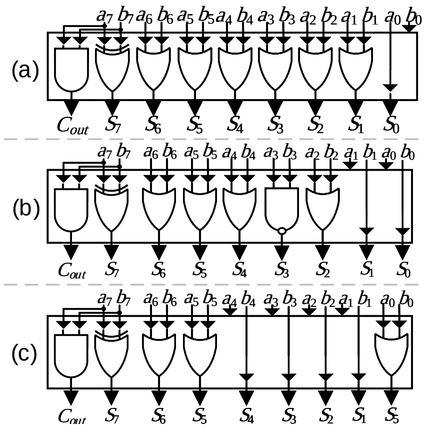


Fig. 15. Normalized MED-Power-Area.



(a)
(b)
(c)
a_7 b_7 a_6 b_6 a_5 b_5 a_4 b_4 a_3 b_3 a_2 b_2 a_1 b_1 a_0 b_0
C_{out} S_7 S_6 S_5 S_4 S_3 S_2 S_1 S_0

Fig. 14. Gate-level implementation of the chosen designs. a) U_1; b) U_12; c) U_146.

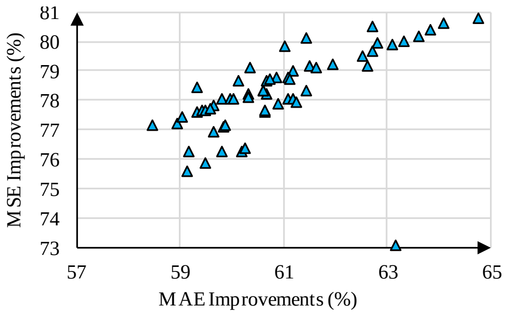


Fig. 16. MSE and MAE Improvements.

[10], [15], and [8]. Moreover, based on the motivational example given in Section 2.2.2, when dealing with different applications, the impact of input patterns should be taken into considerations. Therefore, the third experiment aims at using the first two benefits of the proposed method (i.e., speed up gains and efficient design space exploration) to generate energy-area-efficient approximate adders in an image addition problem. In this experiment, more than two hundred different images were randomly selected from ImageNet [28] database with the keywords ”Nature”, ”Natural”, ”Plants”, ”Flowers”, ”Food”, ”Vegetables”, and ”Salad”.

Similar to the second experiment, power and area constraints of AxMAP are set to an amount slightly lower than those of LOA. The delay is set the delay of a single-bit full adder. The pair of input patterns is set to the average pair of input patterns of all two hundred images from the selected images of ImageNet database. This average pair of input patterns is called ”Centroid” (for this experiment, only two images ,with input patterns similar to the Centroid, were selected for all adders). So, the outputs which perform the best on the addition of the specified input pattern candidate of the two hundred images is chosen, unlike the second experiment in which AxMAP was looking for near optimal adders on uniform input patterns. After about 12 hours, AxMAP was able to generate more than 50 adders that are entirely superior to LOA regarding both circuit, power consumption, area occupation, and application dictated metrics, i.e., MED, PSNR (Peak Signal to Noise Ratio), MSE (Mean Squared Error), and MAE (Mean Absolute Error) for all the given pair of input patterns.

The trade-off between the normalized power-area products and the PSNR improvements of the generated adders over LOA is depicted in Figure 15. As shown in the figure, all adders have better performance over LOA, concerning their output PSNRs, ranging from 73 to 81%. Also, Figure 15 shows reductions in power-area products of outputs over LOA, in range 10 to 50%. To clarify more, it should be mentioned that in a single metric comparison, each of the adders are superior to LOA in terms of power, area, and MED. Since the output designs have better MED when operating on the given input patterns, it is somehow expected to perform better concerning several image metrics such as the MSE and MAE, that their numerical computation is rather MED-like [15]. As can be seen in Figure 16 the MSE and MAE of the generated adders are at least 73% and 58% better than those of LOA, respectively.

4.5 Experiment 4: Testing the input-aware generated adders on different images

The previous experiment showed the benefits generated adders performance regarding circuit characteristics and all the error characteristics for a specified pair of input patterns. However, to show the specialized adders performance in image addition problem, a much more comprehensive comparison has to be accomplished. Therefore, this experiment aims on comparing the image related metrics with those of LOA, when they both employed for a wider range of input images, instead of comparing them for a single image addition problem over the Centroid input pattern. For this experiment, for each of the adders, we randomly choose ten different pair of images from the two hundred images of ImageNet database and performed image addition for both the generated adders and LOA.

Figure 17 shows the PSNR and normalized power-area product diagram of generated adders (more than 50 novel adders). As shown in the figure, in some cases, the generated adders have up to 90% better PSNR than LOA, while having 33% less power-area product (in the best case, the output PSNR is equal to 10.44

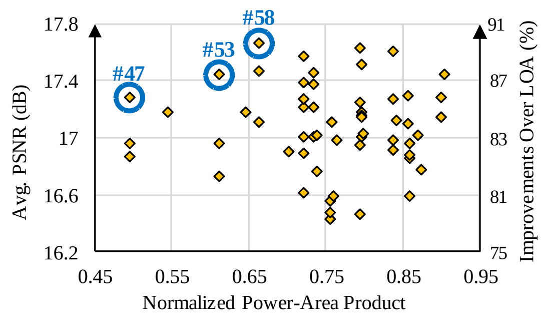


Fig. 17. Average PSNR and Normalized Power-Area Product.

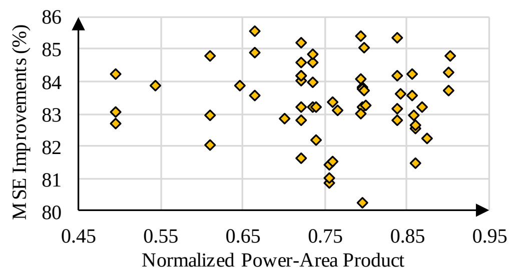


Fig. 18. MSE Improvements and Power-Area Product.

(dB) when employing LOA). Figure 18 and Figure 19 depicts the MSE/MAE power-area product diagrams for output adders. Based on these to figures, the most power area efficient design has almost 50% less power area product than LOA, while improving the MSE and MAE 66% and 84%, respectively.

To further demonstrate the benefits of input-aware circuit generation, the outputs are evaluated for an edge detection (i.e., the Sobel filter) application. We replaced the accurate adders in this filter with the proposed approximate adders (i.e., outputs of AxMAP) and with LOA. For each design ID, we randomly selected an image, and we calculated its corresponding PSNR, MSE, and MAE. Figure 21a, 21b, and 21c show the PSNR, MSE, and MAE of filtered outputs when implemented using the proposed adders and LOA. Based on Figure 21a, concerning the PSNR metric, the proposed adders perform better than LOA ranging from 18% (for design #18) to 85% (for design #25). Based on Figure 21b and 21c, when the filter is implemented with the proposed adders the MSE and MAE are reduced, compared to the LOA-based filter. In worst case, for design #18, the MSE and MAE are 31 and 15% less than those of LOA-based filter. In the best case, for design #25, the outputs has 69 and 46% lower MSE and MAE.

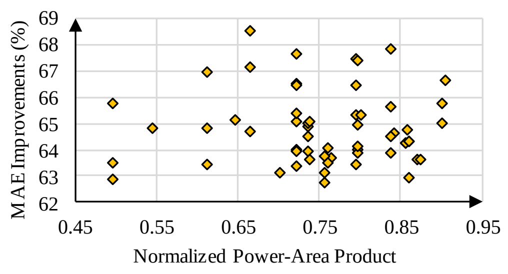


Fig. 19. MAE Improvements and Power-Area Product.

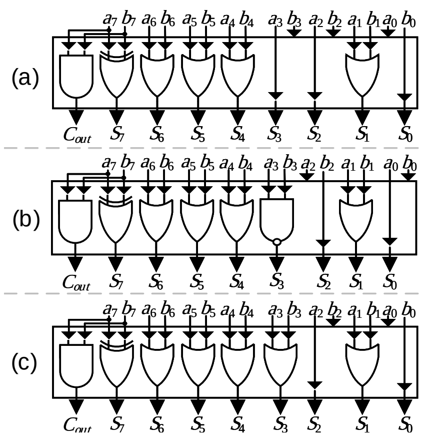
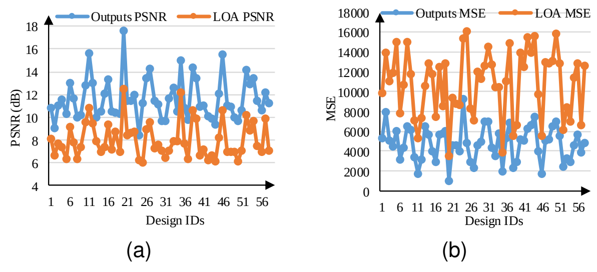


Fig. 20. Gate-level implementation of the chosen designs. (a) N_47; (b) N_53; (c) N_58.



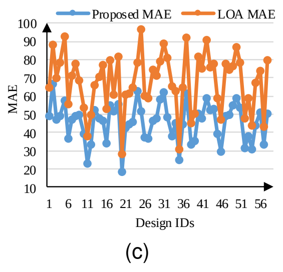


Fig. 21. Comparison of the output designs with LOA in the Sobel filter. (a) PSNR; (b) MSE; (c) MAE.

5 Conclusions

This paper examined a new method for calculating the error metrics (MED and EP) alongside a new framework for automatic generation and design space exploration of energy-area-efficient high speed approximate multi-bit adders. The method for error computation demonstrated more than 150x speed up in comparison with the existing Monte Carlo method. Furthermore, the method was able to accurately compute the MED and EP of adders for any given pair of input patterns. For automatic generation of adder circuits, the paper has presented a publicly available[1] toolchain called AxMAP, that randomly explores the design space and outputs the design with higher accuracy and circuit efficiency regarding state-of-the-art designs.

References

[1] 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.

1. https://github.com/mohrez86/AxMAP

[2] S. Tajasob, M. Rezaalipour, M. Dehyadegari, and M. N. Bojnordi, “Designing efficient imprecise adders using multi-bit approximate building blocks,” in *Proc. International Symposium on Low Power Electronics and Design*, 2018, pp. 13:1–13:6.

[3] T. Yang, T. Ukezono, and T. Sato, “A low-power configurable adder for approximate applications,” in *Proc. 19th International Symposium on Quality Electronic Design*, 2018, pp. 347–352.

[4] T. Yang, T. Ukezono, and T. Sato, “A low-power yet high-speed configurable adder for approximate computing,” in *Proc. IEEE International Symposium on Circuits and Systems*, 2018, pp. 1–5.

[5] M. Osta, A. Ibrahim, H. Chible, and M. Valle, “Inexact arithmetic circuits for energy efficient iot sensors data processing,” in *Proc. IEEE International Symposium on Circuits and Systems*, 2018, pp. 1–4.

[6] W. Xu, S. S. Sapatnekar, and J. Hu, “A simple yet efficient accuracy-configurable adder design,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 6, pp. 1112–1125, 2018.

[7] H. A. F. Almurib, T. N. Kumar, and F. Lombardi, “Inexact designs for approximate low power addition by cell replacement,” in *Proc. Design, Automation & Test in Europe Conference & Exhibition*, 2016, pp. 660–665.

[8] Z. Yang, J. Han, and F. Lombardi, “Transmission gate-based approximate adders for inexact computing,” in *Proc. IEEE/ACM International Symposium on Nanoscale Architectures*, 2015, pp. 145–150.

[9] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, “A low latency generic accuracy configurable adder,” in *Proc. 52nd ACM/EDAC/IEEE Design Automation Conference*, 2015, pp. 1–6.

[10] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, “Low-power digital signal processing using approximate adders,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124–137, 2013.

[11] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi, “Approximate xor/xnor-based adders for inexact computing,” in *Proc. 13th IEEE International Conference on Nanotechnology*, 2013, pp. 690–693.

[12] A. B. Kahng and S. Kang, “Accuracy-configurable adder for approximate arithmetic designs,” in *Proc. Design Automation Conference*, 2012, pp. 820–825.

[13] 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 Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 4, pp. 850–862, 2010.

[14] S. Dutt, S. Dash, S. Nandi, and G. Trivedi, “Analysis, modeling and optimization of equal segment based approximate adders,” *IEEE Transactions on Computers*, vol. 68, no. 3, pp. 314–330, March 2019.

[15] C. Liu, J. Han, and F. Lombardi, “An analytical framework for evaluating the error characteristics of approximate adders,” *IEEE Transactions on Computers*, vol. 64, no. 5, pp. 1268–1281, 2015.

[16] 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.

[17] J. Liang, J. Han, and F. Lombardi, “On the reliable performance of sequential adders for soft computing,” in *2011 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, Oct 2011, pp. 3–10.

[18] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, “Salsa: Systematic logic synthesis of approximate circuits,” in *Proc. Design Automation Conference*, 2012, pp. 796–801.

[19] Y. Wu and W. Qian, “An efficient method for multi-level approximate logic synthesis under error rate constraint,” in *Proc. 53nd ACM/EDAC/IEEE Design Automation Conference*, 2016, pp. 1–6.

[20] A. S. Roy and A. S. Dhar, “A novel approach for fast and accurate mean error distance computation in approximate adders,” 2018, arXiv:1803.08005 [cs.OH].

[21] Y. Wu, Y. Li, X. Ge, Y. Gao, and W. Qian, “An efficient method for calculating the error statistics of block-based approximate adders,” *IEEE Transactions on Computers*, vol. 68, no. 1, pp. 21–38, 2019.

[22] M. K. Ayub, O. Hasan, and M. Shafique, “Statistical error analysis for low power approximate adders,” in *Proc. 54th ACM/EDAC/IEEE Design Automation Conference*, 2017, pp. 1–6.

[23] L. Li and H. Zhou, “On error modeling and analysis of approximate adders,” in *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2014, pp. 511–518.

[24] R. P. Grimaldi, *Discrete and Combinatorial Mathematics: An Applied Introduction, Fifth Edition*. Pearson, 2003.

[25] NanGate, Inc. NanGate FreePDK45 Open Cell Library. [Online]. Available: https://www.silvaco.com/products/nangate/Library_Creator_Platform/index.html?page_id=2325

[26] H. Jiang, J. Han, and F. Lombardi, “A comparative review and evaluation of approximate adders,” in *Proc. 25th Edition on Great Lakes Symposium on VLSI*, 2015, pp. 343–348.

[27] A. Najafi, M. Weisbrich, G. Payá-Vayá, and A. Garcia-Ortiz, “A fair comparison of adders in stochastic regime,” in *Proc. 27th International Symposium on Power and Timing Modeling, Optimization and Simulation*, 2017, pp. 1–6.

[28] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

Morteza Rezaalipour received the M.Sc. degree in computer systems architecture engineering from K. N. Toosi University of Technology, in 2019, under the supervision of Prof. M. Dehyadegari. He is currently a researcher at the K. N. Toosi University of Technology. His research interests are approximate computing, low-power circuits design, and computer architecture.

Mohammad Rezaalipour received the M.Sc. degree in computer engineering - software from Shahid Beheshti University. He is currently pursuing the Ph.D. in computer scinece at the Software Institute, Faculty of Informatics, Università della Svizzera italiana (USI). His research interests include automated program repair and software testing.

Masoud Dehyadegari received his Ph.D. degree from University of Tehran, Tehran, IRAN, in 2013 in computer engineering. He is currently an Assistant Professor of school of computer engineering with the K. N. Toosi University of Technology, Tehran, IRAN. Form September 2011 until December 2012, he was a visiting scholar in University of Bologna, Italy. His research interests include Low-power system design, Network-on-chips,and Multi-Processor System-on-chip.

Mahdi Nazm Bojnordi received the Ph.D. degree in electrical and computer engineering from the University of Rochester, Rochester, NY, USA, in 2016. He is currently an Assistant Professor at the School of Computing, University of Utah, Salt Lake City, UT, USA, where he leads the Energy-Efficient Computer Architecture Laboratory. His current research interests include energy-efficient architectures, low-power memory systems, and the application of emerging memory technologies to computer systems. Dr. Bojnordi received the two IEEE Micro Top Picks Awards, the HPCA 2016 Distinguished Paper Award, and the Samsung Best Paper Award for his research.