An Accurate and Efficient Method to Calculate the Error Statistics of Block-based Approximate Adders

Yi Wu†, You Li†, Xiangxuan Ge†, and Weikang Qian, Member, IEEE

Abstract—Adders are key building blocks of many error-tolerant applications. Leveraging the application-level error tolerance, a number of approximate adders were proposed recently. Many of them belong to the category of block-based approximate adders. For approximate circuits, besides normal metrics such as area and delay, another important metric is the error measurement. Given the popularity of block-based approximate adders, in this work, we propose an accurate and efficient method to obtain the error statistics of these adders. We first show how to calculate the error rates. Then, we demonstrate an approach to get the exact error distribution, which can be used to calculate other error characteristics, such as mean error distance and mean square error.

Index Terms—Approximate computing, Approximate adders, Error rate, Error distribution

1 INTRODUCTION

A approximate circuits implement an approximate version of the target function. They are very attractive for error-tolerant applications, such as image processing, multimedia, and machine learning, since they can trade off accuracy for improvement in circuit area, delay, and power consumption [1].

Given the importance of adders in building many error-tolerant applications, approximate adders have attracted a lot of research effort recently. A number of approximate adders were proposed in literature [2], [3], [4], [5], [6], [7], [8], [9]. Generally speaking, there are two design types. The first type replaces the 1-bit full adders at less significant bit positions by a simpler but inaccurate module. For example, an OR gate is used in the Low-Part-OR adder [2], and an approximate mirror adder is used in [3] to substitute the accurate 1-bit full adder at the lower bit positions. The more significant part is intact. As a result, the reduction in delay and power consumption is limited. Furthermore, this kind of designs could have a high error rate.

The second design type is known as block-based approximate adder [10]. It divides the entire adder into a number of blocks. The calculation of the sum in each block exploits the carry speculation mechanism. It is based on the observation that long carry chain rarely happens in the addition of random inputs. Therefore, the carry chain for calculating each sum bit can be truncated at a middle bit position. Although the carry-in signal for calculating each sum bit could be wrong, the critical path delay and the power consumption are reduced. The majority of the available approximate adders fall into this category. Examples include the Almost Correct Adder [4], the Error Tolerant Adder Type II [5], and the Carry-Skip Approximate Adder [6]. (More details of these approximate adders will be discussed in Section 2.) This type of adder generally has a low error rate. A previous work [10] also showed that it can achieve minimum mean error distance under some conditions. Given its popularity and optimality, we focus on block-based approximate adders in our work.

To measure the performance of an approximate adder, besides the normal metrics such as area, delay, and power consumption, we also need error statistics, including error rate, mean error distance, and mean square error. Although the error statistics of each proposed approximate adder were analyzed by its authors, the method is ad hoc, depending on the structure of the proposed adder. Furthermore, not every error metric is given. For example, for some approximate adders, only error rates were studied, but neither mean error distance nor mean square error was reported.

To address the above problems, three recent works [10], [11], [12] proposed general methods for obtaining important error metrics of block-based approximate adders. [11] proposed an analytical framework to evaluate the error statistics of three types of adders: the Almost Correct Adder, the Equal Segmentation Adder, and the Error Tolerant Adder Type II. However, its results are just estimates and different approaches are applied to evaluate different types of adders. A more general framework was proposed in [10], which can be applied to a wider range of approximate adders. It gives an accurate analysis on the mean error distance, but the error rate and the mean square error are still estimates, not exact results. In some cases, the estimates could be more than 7% away from the accurate values. Also, none of [10] and [11] showed how to obtain the exact error distributions. A recent work [12] provided an accurate method to obtain the exact error distributions. However, its aim is to provide an approach to analyze a more general type of approximate adder, which also includes block-based approximate adder. As a result of making the approach more general, the method sacrifices efficiency and hence needs a long runtime in analyzing adders of large sizes. Given that the block-based approximate adders are one of the most common designs and have good performance, in this work, we specifically target at this type of adders and propose an efficient method to obtain their exact error distributions.

Our method works under the assumption that the inputs to the approximate adder are uniformly distributed. We make this assumption because:

1) Many approximate adders are not just designed for a specific application. To estimate the overall performance of an approximate adder over a range of ap-
plications, it is reasonable to assume the inputs are uniformly distributed.
2) For many specific applications, the inputs are more or less close to uniform distribution.
3) A number of previous works ([2], [5], [6], [7], [10], [11], [13], [14]) in analyzing the error statistics of approximate adders also make this assumption.

Under the assumption of uniform distribution, we first show an accurate and efficient method to calculate the error rate. Using this technique, we further demonstrate an approach to calculate the exact error distribution, by which we can easily obtain other error metrics of interest, such as mean error distance and mean square error. Compared to the previous analytical approaches [10], [11], our method is able to generate the exact error distributions and gives exact error characteristics. Compared to the previous work [12], our method is much faster because it exploits the specific properties on the error patterns of the block-based approximate adders. Indeed, our method achieve the theoretical lower bound on the asymptotic runtime. The proposed method provides an important aid to designers in choosing a proper approximate adder.

In summary, the main contributions of our works are as follows:
1) We propose an accurate and efficient method to obtain
the error rate of the block-based approximate adders.
2) We propose an efficient method to obtain the exact error
distribution of the block-based approximate adders, which can be used to get other error characteristics accurately. As we will show, the asymptotic runtime of our method reaches the theoretical lower bound.
3) We apply our method to obtain the error distributions of several previously proposed approximate adders. The results demonstrate the existence of special patterns on the error distributions of these approximate adders. We give explanation for these special patterns.
4) We demonstrate experimentally the proposed method is much faster and more accurate than the Monte Carlo sampling method to obtain error statistics, especially when the error probability is very small.

The remainder of the paper is organized as follows. Section 2 introduces the general model of the block-based approximate adder and links it to some previously proposed approximate adders. Section 3 discusses some preliminaries. Section 4 and Section 5 show our method to calculate error rate and error distribution, respectively. Section 6 presents the experimental results. Finally, Section 7 concludes the paper.

2 BLOCK-BASED APPROXIMATE ADDERS

In this section, we first overview some previously proposed approximate adders. Then we demonstrate that all of them can be viewed as block-based approximate adder. The adder proposed in [15] and the Almost Correct Adder (ACA) [4] has a structure shown in Fig. 1. Each sum bit is produced by a full adder, which takes a carry-in as input. However, the carry-in is only obtained from l bits before the sum bit instead of all the remaining bits. The Error Tolerant Adder Type II (ETA-II), as shown in Fig. 2, divides the entire n-bit adder into m sub-adders of equal bit length of \( k = n/m \) [6]. The carry-in signal to each sub-adder is produced from the previous k bits by a carry generator, while the carry-in to each carry generator is a logic 0, which essentially truncates the carry chain. Other block-based approximate adders include Speculative Carry Select Adder (SCSA) [13], Error Tolerant Adder Type IV (ETA-IV) [16], and Carry-Skip Approximate Adder (CSAA) [6].

At the behavior level, given the same sub-adder length, SCSA is the same as ETA-II. ETA-IV is similar to ETA-II except that the length of its sub-adder is twice that of its carry generator. CSAA is also similar to ETA-II except that the length of its carry generator is twice that of its sub-adder.

All of the above-mentioned approximate adders can be viewed as block-based approximate adder 10, with a general model shown in Fig. 3. Notice that a similar model was also proposed in [17]. In the model, the number of bits is \( n \). Assume the two addends of an n-bit adder are \( A = a_{n-1} \ldots a_0 \) and \( B = b_{n-1} \ldots b_0 \). The approximate sum is denoted as \( S' = s'_{n-1} \ldots s'_0 \). The carry-out of the approximate adder is denoted as \( c^*_i \). In the model, the sum is divided into a number of blocks which are calculated separately. All the blocks are with the same bit length of \( k \), where \( k \) is a factor of \( n \). Let \( m = n/k \), which represents the number of blocks. In the model, the sum bits in the i-th \((0 \leq i \leq n-1)\) block, \( s_{i+1} \ldots s_{ik} \), are generated by a sub-adder, which takes a speculated carry-in \( c^*_i \). In the ideal case, the carry-in should be produced by all the input bits lower than the position \( ik \). However, for the block-based approximate adder, \( c^*_i \) is produced by a truncated carry generator of length \( l \), as shown in Fig. 3. The carry generator also takes a speculated carry-in \( c_{carry,i} \). For most of the approximate adders, \( c_{carry,i} = 0 \). Thus, in the following analysis, we will assume that \( c_{carry,i} = 0 \), although our analysis is equally applicable to the case where \( c_{carry,i} = 1 \). Note that for all \( 0 \leq i \leq \lfloor l/k \rfloor \), the speculated carry-in \( c^*_i \) is produced by all the remaining input bits and hence, it is always correct. The carry-out \( c^*_i \) of the entire adder is produced by the leftmost sub-adder.

In summary, a block-based approximate adder is characterized by 3 parameters, \( n, k, \) and \( l \), where \( n \) is the adder size, \( k \) is the block size, and \( l \) is the number of bits used in the carry generator. All of the above-mentioned approximate adders are just special cases of this model. For
example, the adder proposed in [15] and ACA correspond to the case where \( k = 1 \). ETA II and SCSA correspond to the case where \( k = l \). ETA-IV corresponds to the case where \( k = 2l \). CSAA corresponds to the case where \( k = 2k \).

Generally speaking, block-based approximate adder can be extended to one with different sub-adder lengths and carry generator lengths for different blocks, which is the one considered in [12]. However, many existing approximate adders have the same sub-adder length and the same carry-generator length for all the blocks, since for this kind of design, no particular block dominates the critical path length. Given this fact, in this work, we target at the block-based approximate adders. For these adders, as we will show in Section 5, our method to obtain the exact error distribution has the lowest asymptotic runtime. However, it should be pointed out that our proposed method can be easily extended to handle the more general situation, at the cost of increasing the asymptotic runtime.

### 3 Preliminaries

In this section, we show some preliminaries that will be used in our later analysis. We assume the inputs \( A \) and \( B \) are uniformly distributed in \([0, 2^n - 1]\) and the carry-in to the entire adder is 0.

#### 3.1 Propagate, Generate, and Kill Signals

For each bit \( i \) (\( 0 \leq i \leq n - 1 \)) in the adder, the propagate, generate, and kill signals of that bit are defined as

\[
p_i = a_i \oplus b_i, \quad g_i = a_i \cdot b_i, \quad k_i = \overline{a_i} \cdot \overline{b_i}.
\]

If \( g_i = 1 \), the carry-out of bit \( i \) is 1 regardless what the carry-in to bit \( i \) is. Similarly, if \( k_i = 1 \), the carry-out of bit \( i \) is always 0. If \( p_i = 1 \), then the carry-in of bit \( i \) propagates to the carry-out of that bit.

For the \( i \)-th (\( 0 \leq i \leq m - 1 \)) block of the adder, we define the group propagate, generate, and kill signal as

\[
P_i = \prod_{j=ik}^{(i+1)k-1} p_j,
\]

\[
G_i = \sum_{j=ik}^{(i+1)k-1} g_j \prod_{d=j+1}^{(i+1)k-1} p_d,
\]

\[
K_i = \sum_{j=ik}^{(i+1)k-1} k_j \prod_{d=j+1}^{(i+1)k-1} p_d.
\]

If \( G_i = 1 \), the carry-out of the \( i \)-th block will always be the correct value of 1 no matter its carry-in is correct or not. Similarly, if \( K_i = 1 \), the carry-out will always be the correct value of 0. Only when \( P_i = 1 \) does the carry-out depend on the carry-in signal, which could be wrong. The probabilities of the above signals being one are

\[
P(P_i = 1) = \frac{1}{2^n},
\]

\[
P(G_i = 1) = \frac{1}{2} - \frac{1}{2^{n+k}},
\]

\[
P(K_i = 1) = \frac{1}{2} - \frac{1}{2^{n+k}}.
\]

#### 3.2 Typical Error Measurement

Typical error measurement of an approximate arithmetic circuit includes error rate, mean error distance, and mean square error.

First, we define the error distance (ED) as the difference of the approximate sum \( S^* \) and the accurate sum \( S \), i.e.,

\[
ED = |S^* - S|.
\]

The error rate (ER) is defined as the percentage of input combinations for which the approximate adder produces a wrong result, i.e., a non-zero error distance. Mathematically, it is calculated as

\[
ER = P(ED \neq 0).
\]

Mean error distance (MED) is the mean value of all the error distances. Mean square error (MSE) are the mean value of the squares of all the error distances. Mathematically, they are calculated as

\[
MED = E[ED] = \sum_{ED \in \Omega} ED_i P(ED_i),
\]

\[
MSE = E[ED^2] = \sum_{ED \in \Omega} ED^2_i P(ED_i),
\]

where \( \Omega \) is the set of all error distances.

### 4 Calculating Error Rate

In this section, we show the method to calculate the error rate. It will be used later to obtain the exact error distribution.

As can be seen in Fig. 3, the result of the approximate adder is correct if and only if all the speculated carry-in \( c_i^* \)’s (\( 0 \leq i \leq m - 1 \)) are correct. To calculate the error rate, we define the event \( D_i \) as the event in which all the speculated carry-ins \( c_i^*, c_{i-1}^*, \ldots, c_0^* \) are correct. We denote the probability of the event \( D_i \) to occur as \( d_i \). Thus, the error rate equals 1 - \( \sum d_i \). In the following, we will derive a recursive formula to calculate \( d_i \). We denote the correct carry-in to the \( i \)-th sub-adder as \( c_i \). Since the recursive formula differs based on whether or not the carry generator length \( l \) is a multiple of the block size \( k \), we will distinguish these two cases.

#### 4.1 Carry Generator Length \( l \) is a Multiple of Block Size \( k \)

Define \( t = \frac{l}{k} \). Then \( t \) represents the number of blocks covered by each carry generator. To illustrate our proposed method, we use \( t = 2 \) as an example. In this case, the carry generator includes 2 blocks of inputs. For \( 0 \leq i \leq 2 \), since all the remaining input bits are used to generate the speculated carry-ins \( c_i^*, c_{i-1}^*, \ldots, c_0^* \), the event \( D_i \) always happens. Thus, \( d_i = 1 \), for all \( 0 \leq i \leq 2 \).

Next we consider \( d_i \) for \( i > 2 \). The event \( D_i \) depends on the inputs from block \( i - 1 \) to 0. Our idea to calculate \( d_i \) is to consider the inputs block by block from block \( i - 1 \) to block 0.
First consider the inputs at block $i-1$. They satisfy either $G_{i-1} = 1$, $K_{i-1} = 1$, or $P_{i-1} = 1$. If the inputs satisfy that $G_{i-1} = 1$, as shown in Fig. 4(a), the speculated carry-in $c_i^* = 1$ is equal to the correct carry-in $c_i$. Thus, the event $D_i$ happens if and only if $c_{i-1}, c_{i-2}, c_{i-3}$ are correct, which means the inputs from block $i-2$ to 2 make the event $D_{i-1}$ happen. Therefore, we have

$$P(D_i, G_{i-1} = 1) = P(G_{i-1})P(D_{i-1}).$$

The same conclusion holds if the inputs at block $i-1$ satisfy that $K_{i-1} = 1$, as shown in Fig. 4(b). Therefore, we have

$$P(D_i, K_{i-1} = 1) = P(K_{i-1})P(D_{i-1}).$$

If the inputs at block $i-1$ satisfy $P_{i-1} = 1$, then we further consider the inputs at block $i-2$. We also distinguish them into three cases: $G_{i-2} = 1$, $K_{i-2} = 1$, and $P_{i-2} = 1$. In the case where $G_{i-2} = 1$ (shown in Fig. 4(c)) and the case where $K_{i-2} = 1$ (shown in Fig. 4(d)), since the carry generator covers 2 blocks of inputs, the speculated carry-ins $c_i^*$ and $c_{i-1}^*$ are equal to the correct carry-ins $c_i$ and $c_{i-1}$, respectively. Thus, the event $D_i$ happens if and only if $c_{i-1}, c_{i-2}, c_{i-3}$ are correct, which means the inputs from block $i-3$ to 0 make the event $D_{i-1}$ happen. Therefore, we have

$$P(D_i, P_{i-1} = 1, G_{i-1} = 1) = P(P_{i-1})P(G_{i-2})P(D_{i-2}),$$

$$P(D_i, P_{i-1} = 1, K_{i-1} = 1) = P(P_{i-1})P(K_{i-2})P(D_{i-2}).$$

If the inputs at blocks $i-2$ and $i-3$ satisfy none of the above cases, then we must have $P_{i-2} = P_{i-3} = 1$. Now we further consider the inputs at block $i-3$. We distinguish the following three cases:

1) The inputs satisfy that $G_{i-3} = 1$, as shown in Fig. 4(e). In this case, the correct carry-ins $c_i = c_{i-1} = c_{i-2} = 1$. However, the speculated carry-in $c_i^*$ is 0, since it is produced by a carry generator that covers inputs at blocks $i-1$ and $i-2$ and that carry generator propagates the speculated carry-in to the carry generator, which is assumed to be 0. Since $c_i^* \neq c_i$, the event $D_i$ cannot happen in this case. Therefore, we have

$$P(D_i, P_{i-1} = P_{i-2} = G_{i-3} = 1) = 0.$$

2) The inputs satisfy that $K_{i-3} = 1$, as shown in Fig. 4(f). In this case, the correct carry-ins $c_i = c_{i-1} = c_{i-2} = 0$. By the same argument used in Case 1, the speculated carry-in $c_i^*$ must be 0. Since each carry generator covers two blocks of inputs, the speculated carry-in $c_{i-1}^*$

$$c_{i-1}^* = c_j = 0 \text{ for } j = i-1, i-2.$$ Thus, the event $D_i$ happens if and only if $c_{i-3}^*, \ldots, c_0^*$ are correct, which means the inputs from block $i-4$ to 0 make the event $D_{i-3}$ happen. Therefore, we have

$$P(D_i, P_{i-1} = P_{i-2} = K_{i-3} = 1) = P(P_{i-1})P(P_{i-2})P(K_{i-3})P(D_{i-3}).$$

3) The inputs satisfy that $P_{i-3} = 1$. In this case, we continue to look at the inputs at block $i-4$.

We continue the above analysis. By the same reasoning used for the case where $P_{i-1} = P_{i-2} = G_{i-3} = 1$, we have that for any $3 < j \leq i$, if the inputs from block $i-1$ to block $i-j$ satisfy that $P_{i-1} = \cdots = P_{i-j+1} = G_{i-j} = 1$, the event $D_i$ cannot happen, since $c_{i} = 0 \neq c_i = 1$, i.e.,

$$P(D_i, P_{i-1} = P_{i-2} = \cdots = P_{i-j+1} = G_{i-j}) = 0.$$ (10)

On the other hand, if the inputs satisfy that $P_{i-1} = \cdots = P_{i-j+1} = 1$, the event $D_i$ happens if and only if the inputs from block $i-j-1$ to 0 make the event $D_{i-j}$ happen. Therefore, we have

$$P(D_i, P_{i-1} = P_{i-2} = \cdots = P_{i-j+1} = K_{i-j} = 1) = P(P_{i-1})P(P_{i-2}) \cdots P(P_{i-j+1})P(K_{i-j})P(D_{i-j}).$$

Finally, there is a remaining input case which satisfies that $P_{i-1} = \cdots = P_{i-j+1} = G_{i-j} = 1$ and $P_{i-j+2} = 0$. In this case, the speculated carry-ins are $c_{i-1}^* = \cdots = c_0^* = 0$ and the correct carry-ins are $c_{i-1} = \cdots = c_0 = 0$. Thus, the event $D_i$ happens. Therefore, we have

$$P(D_i, P_{i-1} = P_{i-2} = \cdots = P_{i-j+1} = G_{i-j} = 1) = P(P_{i-1})P(P_{i-2}) \cdots P(P_{i-j+1}).$$

Notice the probability that the event $D_i$ occurs can be calculated as

$$P(D_i) = P(D_i, G_{i-1} = 1) + P(D_i, K_{i-1} = 1) + P(D_i, P_{i-1} = 1, G_{i-1} = 1)$$

$$+ P(D_i, P_{i-1} = 1, G_{i-1} = 1) + \cdots + P(D_i, P_{i-j+1} = 1)$$

$$+ P(D_i, P_{i-j+1} = 1, K_{i-j} = 1).$$

Given Eqs. (4)-(12), we can calculate $d_i$ as follows:

$$d_i = P(D_i) = P(G_{i-1})P(D_{i-1}) + P(K_{i-1})P(D_{i-1})$$

$$+ P(P_{i-1})P(G_{i-2})P(D_{i-2}) + P(P_{i-1})P(K_{i-2})P(D_{i-2})$$

$$+ P(P_{i-1})P(P_{i-2})P(K_{i-3})P(D_{i-3})$$

$$+ P(P_{i-1})P(P_{i-2})P(P_{i-3})P(K_{i-4})P(D_{i-4})$$

$$+ \cdots + P(P_{i-1})P(P_{i-2}) \cdots P(P_{i})P(K_0)P(D_0)$$

$$+ P(P_{i-1})P(P_{i-2}) \cdots P(P_{i})P(D_0).$$

$$= P(P_{i-1}) \cdots P(P_0) + \sum_{j=1}^{t} P(P_{i-1}) \cdots P(P_{i-j+1})P(G_{i-j})d_{i-j}$$

$$+ \sum_{j=1}^{t} P(P_{i-1}) \cdots P(P_{i-j+1})P(K_{i-j})d_{i-j}.$$
The above equation gives a recursive way to calculate \( d_i \). The values of \( P(D_i), P(G_i), P(K_i) \) are calculated by Eq. (1), (2), and (3), respectively. The time complexity to obtain \( d_{m-1} \) and the resultant error rate is \( O(m^2) \).

4.2 Carry Generator Length \( l \) is not a Multiple of Block Size \( k \)

Define \( t = \lfloor \frac{l}{k} \rfloor \). Each carry speculative chain is composed of \( t \) blocks of bit length \( k \) and a remaining block of bit length \( k' = l - tk \). Note that \( 0 < k' < k \).

For each block of \( k \) bits, we divide it into the left group of \( k' \) bits and the right group of \( k - k' \) bits. The major difference in the analysis here is that we need to consider the propagate/generate/kill state of both the left group and the right group in a block. For the left group of \( k' \) bits in the \( i \)-th (\( 0 \leq i \leq m-1 \)) block of the adder, we define its group propagate, generate, and kill signal as

\[
PL_i = \prod_{j=i(1+k)+1}^{i(1+k)-1} P_j,
\]

\[
GL_i = \sum_{j=i(1+k)+1}^{i(1+k)-1} k_j \prod_{j=i(1+k)+1}^{i(1+k)-1} P_d.
\]

\[
KL_i = \sum_{j=(i+1)k-k}^{i(1+k)-1} k_j \prod_{j=(i+1)k+1}^{i(1+k)-1} P_d.
\]

The probabilities of the above signals being one are

\[
P(PL_i) = P(GL_i) = 1 - \frac{1}{2^{k_t}}.
\]

\[
P(KL_i) = P(KL_i) = 1 - \frac{1}{2^{k_t}}.
\]

Similarly, we define the group propagate, generate, and kill signal of the right group of \( k - k' \) bits of the \( i \)-th block. These signals are denoted as \( PR_i, GR_i, \) and \( KR_i \) respectively. Their probabilities of being one are calculated similarly as above.

To illustrate the method to calculate \( d_i \), we also use \( t = 2 \) as an example. For \( 0 \leq i \leq 2 \), it is not hard to see that \( d_1 = 1 \). Thus, we only focus on \( i > 2 \).

The basic idea to obtain the probability \( d_i \) is same as what we use to handle the case where \( l \) is a multiple of \( k \), i.e., examining inputs block by block from block \( i-1 \) to block 0. For the inputs satisfying either (1) \( G_{i-1} = 1 \), (2) \( K_{i-1} = 1 \), (3) \( P_{i-1} = G_{i-1} = 2 \), or (4) \( P_{i-1} = K_{i-1} = 1 \), the conclusions are the same as what we have when \( l = 2k \).

Fig. 4(a)-(d) show the correct carry-ins and the speculated carry-ins for four different cases under the situation that \( l = 2k + k' \) where \( 0 < k' < k \). (a) \( P_{i-1} = P_{i-2} = GL_{i-3} = 1 \); (b) \( P_{i-1} = P_{i-2} = KL_{i-3} = 1 \); (c) \( P_{i-1} = P_{i-2} = PL_{i-3} = GR_{i-3} = 1 \); (d) \( P_{i-1} = P_{i-2} = PL_{i-3} = KR_{i-3} = 1 \).

Fig. 5: The speculated carry-ins and the correct carry-ins for different input cases under the situation that \( l = 2k + k' \) where \( 0 < k' < k \). (a) \( P_{i-1} = P_{i-2} = GL_{i-3} = 1 \); (b) \( P_{i-1} = P_{i-2} = KL_{i-3} = 1 \); (c) \( P_{i-1} = P_{i-2} = PL_{i-3} = GR_{i-3} = 1 \); (d) \( P_{i-1} = P_{i-2} = PL_{i-3} = KR_{i-3} = 1 \).

if the inputs from block \( i - 4 \) to 0 make the event \( D_{i-3} \) happen. Therefore, we have

\[
P(D_i, P_{i-1} = P_{i-2} = GL_{i-3} = 1) = P(P_{i-1})P(P_{i-2})P(GL_{i-3})P(D_{i-3}).
\]

(14)

3) The inputs satisfy that \( PL_{i-3} = GR_{i-3} = 1 \), as shown in Fig. 5(c). In this case, the correct carry-\( i \)'s \( c_i = c_{i-1} = c_{i-2} = 1 \). However, the speculated carry-in \( c_i = 0 \), since it is produced by a carry generator that covers inputs at block \( i-1 \), block \( i-2 \), and the left group of block \( i-3 \) (see Fig. 4). and that carry generator propagates a 0. Since \( c_i \neq c_j \), the event \( D_i \) cannot happen. Therefore, we have

\[
P(D_i, P_{i-1} = P_{i-2} = PL_{i-3} = GR_{i-3} = 1) = 0.
\]

(15)

4) The inputs satisfy that \( PL_{i-3} = KR_{i-3} = 1 \). In this case, as shown in Fig. 5(d), \( c_j = 0 \) for \( j = i-1, i-2, i-3 \). Thus, the event \( D_i \) happens if and only if the inputs from block \( i-4 \) to 0 make the event \( D_{i-3} \) happen. Therefore, we have

\[
P(D_i, P_{i-1} = P_{i-2} = PL_{i-3} = KR_{i-3} = 1) = P(P_{i-1})P(P_{i-2})P(PL_{i-3})P(KR_{i-3})P(D_{i-3}).
\]

(16)

5) The inputs satisfy that \( P_{i-3} = 1 \). In this case, we need to continue checking the inputs at block \( i-4 \).

Now we consider the remaining case where \( P_{i-1} = P_{i-2} = P_{i-3} = 1 \). We further check the inputs at block \( i-4 \). Similarly, they can be divided into the five cases as shown above. The situations corresponding to the first four cases are shown in Fig. 5(e)-(h), respectively. Since \( P_{i-1} = P_{i-2} = P_{i-3} = 1 \) and each carry generator covers two blocks of inputs plus the left group of the third block, the speculated carry-in \( c_i = 0 \). In Case 1 (i.e., \( GL_{i-4} = 1 \)
and Case 3 (i.e., $PL_{i-4} = GR_{i-4} = 1$), since the correct carry-in $c_i = 1 \neq c_i^*$, the event $D_i$ cannot happen. Therefore we have

$$P(D_i, P_{i-1} = P_{i-2} = P_{i-3} = GL_{i-4} = 1) = 0.$$  \hfill (17)

$$P(D_i, P_{i-1} = P_{i-2} = P_{i-3} = PL_{i-4} = GR_{i-4} = 1) = 0.$$  \hfill (18)

In Case 2 (i.e., $KL_{i-4} = 1$) and Case 4 (i.e., $PL_{i-4} = KR_{i-4} = 1$), $c_j = c_j^*$ for $j = i-1, \ldots, i-4$. Thus, the event $D_i$ happens if and only if the inputs from block $i-5$ to 0 make the event $D_{i-4}$ happen. Therefore we have

$$P(D_i, P_{i-1} = P_{i-2} = P_{i-3} = KL_{i-4} = 1) = P(P_{i-1}) P(P_{i-2}) P(P_{i-3}) P(KL_{i-4}) P(D_{i-4}).$$  \hfill (19)

$$P(D_i, P_{i-1} = P_{i-2} = P_{i-3} = PL_{i-4} = KR_{i-4} = 1) = P(P_{i-1}) P(P_{i-2}) P(P_{i-3}) P(PL_{i-4}) P(D_{i-4}).$$  \hfill (20)

In Case 5, the inputs at block $i-4$ satisfy that $P_{i-4} = 1$, we continue analyzing the inputs of the next block in the same way.

By the same reasoning used for the case where $P_{i-1} = P_{i-2} = P_{i-3} = 0$, we have that for any $4 < j \leq i$, if the inputs from block $i-j$ to block $i-j+1$ satisfy either $P_{i-j} = \cdots = P_{i-j+1} = GL_{i-j} = 1$ or $P_{i-j} = \cdots = P_{i-j+1} = PL_{i-j} = GR_{i-j} = 1$, the event $D_i$ cannot happen. Therefore we have

$$P(D_i, P_{i-1} = P_{i-2} = P_{i-3} = KL_{i-4} = 1) = P(P_{i-1}) P(P_{i-2}) P(P_{i-3}) P(KL_{i-4}) P(D_{i-4}).$$

$$P(D_i, P_{i-1} = P_{i-2} = P_{i-3} = PL_{i-4} = KR_{i-4} = 1) = P(P_{i-1}) P(P_{i-2}) P(P_{i-3}) P(PL_{i-4}) P(D_{i-4}).$$

In the above equation gives a recursive way to calculate $d_i$. The time complexity to obtain $d_{m-1}$ and the resultant error rate $O(m^2)$.

## 5 Obtaining Error Distribution

In this section, we show our method to obtain the accurate error distribution. For a given input combination, the output error is defined as $err = c_0 s_{n-1} \ldots s_0 - c_{0} s_{n-1} \ldots s_0$, where $c_0, s_{n-1}, \ldots, s_0$ are the correct outputs of the adder. By the definition, the error distance of an error is equal to the absolute value of the error. The error distribution is the probability distribution of the error distance. Similar as characterizing the error rate, we will discuss based on whether or not the carry generator length $l$ is a multiple of the block size $k$ in Section 5.1 and 5.2 respectively. Finally, we will discuss the time complexity of our method in Section 5.3.

### 5.1 Carry Generator Length $l$ is a Multiple of Block Size $k$

#### 5.1.1 Error Pattern and Probability

We define $t = l / k$. In this section, we first analyze the error pattern, which is the binary representation of the error distance. The questions we ask are: 1) when does an error happen? 2) if an error happens, what does its error pattern look like?

To analyze the error pattern, we divide the $m$ blocks into a number of partitions. Each partition is composed of two sequences of blocks, where all the blocks in the left sequence have their group propagate signals as 0 and all the blocks in the right sequence have their group propagate signals as 1. For the leftmost (rightmost) partition, it is possible that all of its blocks have their group propagate signals as 1 (0). Fig. 6 gives an example of 10 blocks divided into 4 partitions.

![Fig. 6: An example of 10 blocks divided into 4 partitions.](image)

The above equation gives a recursive way to calculate $d_i$. The time complexity to obtain $d_{m-1}$ and the resultant error rate is $O(m^2)$.

### 5.1.2 Error Pattern and Probability

We define $t = l / k$. In this section, we first analyze the error pattern, which is the binary representation of the error distance. The questions we ask are: 1) when does an error happen? 2) if an error happens, what does its error pattern look like?

To analyze the error pattern, we divide the $m$ blocks into a number of partitions. Each partition is composed of two sequences of blocks, where all the blocks in the left sequence have their group propagate signals as 0 and all the blocks in the right sequence have their group propagate signals as 1. For the leftmost (rightmost) partition, it is possible that all of its blocks have their group propagate signals as 1 (0). Fig. 6 gives an example of 10 blocks divided into 4 partitions.

![Fig. 6: An example of 10 blocks divided into 4 partitions.](image)

We now study the approximate sum in each partition. Suppose the partition starts at block $i_b$ and ends at block $i_e$ ($i_b > i_e$). The blocks $i_b, i_b - 1, \ldots, i_e$ ($i_e < i_m \leq i_b$) have their group propagate signals as 0 and the blocks $i_m - 1, \ldots, i_e$ have their group propagate signals as 1. We have the following two claims on errors in the partition.

**Lemma 1**

There exists an error in the approximate sum of a partition if and only if $G_{i_m-1} = 1$ and $i_m - i_e \geq t$.

**Proof:** "if" part: The carry speculative chain for the carry-in $c^{\ast}_{i_e}$ is composed of the bits in blocks $i_m - 1, \ldots, i_m - t$. Since $P_{i_m-1} = \cdots = P_{i_m-t} = 1$, we have $c_{i_m-1}^* = 0$. However, because $P_{i_m-1} = \cdots = P_{i_e} = 1$ and $G_{i_m-1} = 1$, the correct carry-in $c_{i_m-1} = 1$. Thus, the sum at block $i_m$ is incorrect.

"only if" part: we prove by contradiction. If the condition is not satisfied, then there are two cases: (1) $K_{i_m-1} = 1$ and (2) $G_{i_m-1} = 1$ and $i_m - i_e < t$. For Case 1, the speculated carry-ins $c_{i_m-1}^*, \ldots, c_{i_e}^*$ are all equal to the correct value of 0. For Case 2, the speculated carry-ins $c_{i_m-1}^*, \ldots, c_{i_e}^*$ are all equal to the correct value of 1. For both cases, since $P_{i_m-1} = \cdots = P_{i_m} = 0$, the speculated carry-ins $c_{i_b}, \ldots, c_{i_m-1}$ are
Given Eq. (22), we have that the inputs $s(i−2)k−1 \cdots s(j−2)k$ are all correct and hence, the approximate sum from block $i$ to block $(i−e+t)$ is smaller than the correct sum by 1.

![Block diagram](image)

**Theorem 1**

A 1 in an error pattern can only occur at the bit position $i_k$, where $i$ is an integer. It occurs at the bit position $i_k$ if and only if $P_{i−1} = \cdots = P_{i−t} = 1$ and $G_{i−t−1} = 1$.

Based on the above theorem, we also have the following corollary.

**Corollary 1**

If there are two adjacent 1’s in an error pattern, where the left one is at position $i_k$ and the right one is at position $j_k$, then $j < i − t$.

**Proof:** Since a 1 occurs at the bit position $i_k$ of the error pattern, by Theorem 1 we have $G_{i−1} = 1$. As a result, 1 cannot occur at bit positions $(i−1)k, \ldots, (i−t)k$, because otherwise, due to Theorem 1, we have $P_{i−t−1} = 1$, which contradicts with $G_{i−t−1} = 1$. Therefore, the next 1 in the error pattern must be at the bit position $j_k$, where $j < i − t$.

Now we study the following question: if there is a 1 at position $j_k$ of an error pattern, under what situation does the next 1 on the left of $j_k$ appear at a position $i_k$ where $i > j + t$? Fig. 8 illustrates this problem using an example with $k = 4$ and $t = 2$. As shown in the figure, in the error pattern, there is a 1 at bit position $j_k$ and the next 1 is at bit position $i_k$. Since the correct carry-in to the bit $j_k$ is 1, the correct sum from block $i − 3$ to block $j$ is

$$s(i−2)k−1 \cdots s(j−2)k = a(i−2)k−1 \cdots a jk + b(i−2)k−1 \cdots b jk + 1. \tag{21}$$

where $a(i−2)k−1 \cdots a jk$ and $b(i−2)k−1 \cdots b jk$ denote the inputs from block $i − 3$ to block $j$.

On the other hand, as shown in Fig. 8, except the last bit of block $j$, all the other bits in blocks $i−3, \ldots, j$ of the error pattern $|err|$ are 0. Therefore, the correct sum from block $i − 3$ to $j$ is larger than the approximate sum by 1. Given Eq. (21), the approximate sum is

$$s(i−2)k−1 \cdots s j k = a(i−2)k−1 \cdots a jk + b(i−2)k−1 \cdots b jk. \tag{22}$$

Furthermore, the approximate sum from block $i − 3$ to $j$ is equal to the sum produced by an approximate adder of the same type with $(i − j − 2)$ blocks, where the inputs are $a(i−2)k−1 \cdots a jk$ and $b(i−2)k−1 \cdots b jk$. For clarity, we call this approximate adder **imaginary approximate adder**.

Given Eq. (22), we have that the inputs $a(i−2)k−1 \cdots a jk$ and $b(i−2)k−1 \cdots b jk$ should make the imaginary approximate adder produce the correct sum. This means all the speculated carry-ins $c_{i−j−3}^1, \ldots, c_0^1$ of the imaginary adder should
be correct. In other words, the inputs at blocks $i - 4, \ldots, j$ of the adder in Fig. 8 should make the Event $D_{i - j - 3}$ defined in Section 4 happen. For a general $t$, the inputs at blocks $i - t - 2, \ldots, j$ should make the Event $D_{i - t - j - 1}$ happen.

Therefore, given that there is a 1 at position $j k$ of an error pattern, the next 1 appears at a position $i k$ where $i > j + t$ if and only if $P_{i - t} = \cdots = P_{i - 1} = 1$, $G_{i - t - 1} = 1$, and the inputs at blocks $i - t - 2, \ldots, j$ make the event $D_{i - t - j - 1}$ happen.

The probability is

$$e_{i, j} \triangleq P(P_{i - 1}) \cdots P(P_{i - j})P(G_{i - t - 1})d_{i - t - j - 1}$$

(23)

The rightmost position in the error pattern where a 1 can occur is the position $(t + 1)k$. The probability that the rightmost 1 is at a bit position $i k$ $(i \geq t + 1)$ is

$$e_{i, 0} \triangleq P(P_{i - 1}) \cdots P(P_{i - j})P(G_{i - t - 1})d_{i - t - j - 1}$$

The leftmost position in the error pattern where a 1 can occur is the position $(m - 1)k$. Given that there is a 1 at position $i k$ $(i \leq m - 1)$, there is no 1 on the left of position $i k$ if and only if the inputs at blocks $m - 2, \ldots, i$ make the event $D_{m - i - 1}$ happen. The probability is $d_{m - i - 1}$.

Given the above discussion, we finally get the following claim on error pattern and probability.

**Theorem 2**

A $m k$-bit binary representation is a possible error pattern if and only if there exist integers $t + 1 \leq i_{1} < i_{2} < \cdots < i_{r} \leq m - 1$ such that for all $1 \leq j \leq r - 1$, we have $i_{j + 1} - i_{j} > t$ and the ones in the binary representation appear at bit positions $i_{1}k, \ldots, i_{r}k$. The probability of that error pattern is

$$d_{m - i_{r} - 1} \cdot \prod_{j = 1}^{r} e_{i_{j}, i_{j} - 1}$$

where $i_{0} = 0$.

**5.1.2 Algorithm to Obtain Error Distribution**

Using Theorem 2, we can enumerate all possible error patterns and calculate their probabilities. This gives us a method to obtain the error distribution.

However, the enumeration-based method can be further optimized by saving common multiplications of probabilities and common additions of error components appeared in the calculation. We propose a divide-and-conquer method to do this. The idea is to grow a partial error pattern and its probability into the complete error pattern and the probability. The procedure uses a recursive helper function $ED(i, j, ePar, pPar)$ shown in Algorithm 1. The argument $i$ refers to the bit position $i k$, which is under check and can potentially have a 1. $j$ refers to the bit position $j k$, which is the nearest bit position on the right of $i k$ in the error pattern that has a 1. $ePar$ is the partial error distance and $pPar$ is the partial probability.

When checking the block $i$, we have the following two cases:

1) There is a 1 at the bit position $i k$ in the error pattern. Then, the nearest bit position on the left of $i k$ that could have a 1 is $(i + t + 1)k$. We continue to check that bit position. The error magnitude $2^{i k}$ is added to the partial error distance and the partial probability is multiplied with the value $e_{i, j}$. This is shown in Line 6 of Algorithm 1.

2) There is no 1 at the bit position $i k$ in the error pattern. Then, we further check the bit position $(i + 1)k$. The partial error distance and probability do not change. This is shown in Line 7 of Algorithm 1.

Algorithm 1 $ED(i, j, ePar, pPar)$: a recursive helper function to obtain the error distribution.

1: if $i \geq m$ then
2: $pPar = pPar \cdot d_{m - i - 1}$;
3: Print out $ePar$ and $pPar$;
4: return;
5: end if
6: $ED(i + t + 1, i, ePar + 2^{i k}, pPar \cdot e_{i, j})$;
7: $ED(i + 1, j, ePar, pPar)$;
8: return;

When $i \geq m$, $ePar$ becomes the complete error distance. The probability of $ePar$ should be $pPar$ multiplied by $d_{m - i - 1}$, which accounts for the probability that there is no 1 on the left of position $j k$ in the error pattern. The initial function call is $ED(t + 1, 0, 0, 1)$, because the rightmost position in an error pattern where a 1 can occur is the position $(t + 1)k$, the partial error distance is 0, and the partial probability is 1.

**5.2 Carry Generator Length $l$ is not a Multiple of Block Size $k$**

This situation is slightly more complicated. However, the overall analysis flow is similar to the case where $l$ is a multiple of $k$. Similar as Section 4.2, we define $t = \left\lfloor \frac{l}{k} \right\rfloor$ and $l' = l - t k$. In this situation, Lemma 1 is changed to the following one.

**Lemma 3**

There exists an error in the approximate sum of a partition if and only if either of the following two events happens:

1) $PL_{i_{n - 1}} = GR_{i_{n - 1}} = 1$ and $t_{m} - t_{e} \geq t$, where $PL_{i}$ and $GR_{i}$ are defined in Section 4.2.
2) $GL_{i_{n - 1}} = 1$ and $t_{m} - t_{e} \geq t + 1$.

Proof: “if” part: If the event 1 happens, then the speculated carry-in $c_{i, t + 1}^{n} = 0$, because the carry generator for the carry-in $c_{i, t}^{n}$ covers blocks $i_{e} - t_{e} - 1, \ldots, i_{m}$ plus the left group of block $i_{e} - 1$, and their group propagate signals are all 1. On the other hand, the correct carry-in $c_{i, t + 1}^{n} = 1$. Thus, the sum at block $(i_{e} + t)$ is incorrect. By a similar argument, if the event 2 happens, then the speculated carry-in $c_{i, t + 1}^{n} = 0$, while the correct carry-in $c_{i, t + 1}^{n} = 1$. Thus, the sum at block $(i_{e} + t + 1)$ is incorrect.

“only if” part: we prove by contradiction. If neither of the two events happens, then there are four cases: (1) $KL_{i_{n - 1} - 1} = 1$, (2) $PL_{i_{n - 1} - 1} = GR_{i_{n - 1} - 1} = 1$, (3) $PL_{i_{n - 1} - 1} = GR_{i_{n - 1} - 1} = 1$ and $i_{m} - t_{e} \leq t - 1$, and (4) $GL_{i_{n - 1} - 1} = 1$ and $i_{m} - t_{e} \leq t$. For all cases, the speculated carry-ins $c_{i}^{n}, \ldots, c_{i}^{n}$ are all correct and hence, the approximate sum of the partition is correct.

As a result, Theorem 1 is changed to the following one.

**Theorem 3**

A 1 in an error pattern can only occur at the bit position $i k$, where $i$ is an integer. It occurs at the bit position $i k$ if and only if either of the following two events happens:

1) $P_{1} = \cdots = P_{i - 1} = 1, PL_{i - t - 1} = 1$, and $GR_{i - t - 1} = 1$.
2) $P_{1} = \cdots = P_{i - t - 1} = 1$ and $GL_{i - t - 1} = 2$.

Given the above theorem, we can see that if two adjacent $1$’s in an error pattern are at positions $i k$ and $j k$ $(i > j)$, respectively, then $i - j > t$. For the case where $j$ is not a multiple of $k$, we still use $e_{i, j}$ to denote the probability that given there is a 1 at position $j k$ in an error pattern, the next 1 appears at a position $i k$, where $i > j + t$. However, to
obtain $e_{i,j}$, we need to distinguish between $i = j + t + 1$ and $i > j + t + 1$.

Given that there is a 1 at position $jk$ in an error pattern, the next 1 appears at a position $ik$ where $i = j + t + 1$ if and only if either of the following two events happens:

1) $P_{i-1} = \cdots = P_{i-t} = 1, PL_{i-t-1} = 1, GR_{i-t-1} = 1$, and the inputs at blocks $i-t-2, \ldots, j$ make the event $D_{i-t-j-2}$ happen.

2) $P_{i-1} = \cdots = P_{i-t} = 1, GL_{i-t-2} = 1$, and the inputs at blocks $i-t-3, \ldots, j$ make the event $D_{i-t-j-2}$ happen.

The probability that one of the above two events happens is

$$e_{i,j} \triangleq P(P_{i-1}) \cdots P(P_{i-t})P(PL_{i-t-1})P(GR_{i-t-1})d_{i-t-j-2}.$$

The rightmost position in an error pattern where a 1 can occur is the position $(t+1)k$. It can be shown that the probability that the rightmost 1 is at the position $(t+1)k$ is

$$e_{i+1,0} \triangleq P(P_{1}) \cdots P(P_{j})P(PL_{0})P(GR_{0}),$$

while the probability that the rightmost 1 is at the position $ik$ ($i > t + 1$) is

$$e_{i,0} \triangleq P(P_{i-1}) \cdots P(P_{i-t})P(PL_{i-t-1})P(GR_{i-t-1})d_{i-t-j-1} + P(P_{i-1}) \cdots P(P_{i-t})P(GL_{i-t-2})d_{i-t-2}.$$

The probability that there is a 1 at position $ik$ ($i \leq m-1$) in the error pattern, there is no 1 on the left of position $ik$ is still $d_{n-1-i}$, the same value as we have for the case where $l$ is a multiple of $k$.

For the case where $l$ is not a multiple of $k$, Theorem 2 still holds, which characterizes the error pattern and probability. The only difference is that the values $e_{i,j}$ are replaced with the new $e_{i,j}$ obtained in this section. Finally, the procedure to obtain the error distribution is similar to that used for the case where $l$ is a multiple of $k$. Again, the only change is to use the new values of $e_{i,j}$ obtained in this section.

5.3 Time Complexity Analysis

![Fig. 9: Function calls of Algorithm 1 for a 32-bit CSAA with $k = 4$ and $l = 8$.](image)

As shown in Algorithm 1, the time complexity of the proposed method is proportional to the number of calls of the function $ED$. Since $ED$ is a recursive function and it calls itself twice in each function call, the process of the function calls can be modeled as a binary tree. For example, the binary tree representing the function calls for a 32-bit CSAA with $k = 4$ and $l = 8$ is shown in Fig. 9. Each pair $(x, y)$ in this figure represents one function call of the form $ED(i, j, ePar, pPar)$, where $x$ is equal to $i$ and $y$ is equal to $j$. The internal nodes in this binary tree are represented by square nodes, while the leaf nodes are represented by round nodes. At the leaf, there is no further recursive function call, since $i \geq m = n/k = 32/4 = 8$. The vector under each leaf node represents the error pattern produced at that leaf. A number $j$ in the vector indicates that there is a 1 at the bit position $jk$ of the error pattern. For example, the vector $[6,3]$ represents an error pattern with 1s at bit positions 24 and 12. Note that the rightmost leaf corresponds to an all-zero error pattern. Thus, the vector is $\emptyset$. We can see that the number of error patterns is equal to the number of leaves in the binary tree. Denote the number of error pattern as $N$. Then the total number of nodes in the binary tree is $2N - 1$.

Since each call of $ED$ corresponds to a node in the binary tree, the runtime of the proposed method is proportional to $2N - 1$, which implies that the time complexity is $O(N)$. On the other hand, any algorithm for obtaining the error distribution must have runtime complexity at least $\Omega(N)$, since it must produce all the error patterns and the associated probabilities. Therefore, the proposed algorithm achieves the theoretical lower bound on the asymptotic runtime.

To further quantify the runtime, we analyze the number of error patterns for a given block-based approximate adder. We show how we derive the number for a block-based approximate adder using an example. Consider an adder with $n = 32$, $k = 4$, and $l = 4$. The number of blocks $m = n/k = 8$ and $t = l/k = 1$. The following is a list of all the non-zero error patterns, represented using the vector representation shown in Fig. 9:

$$[7, 6, 5, 4, 3, 2, 1],$$

Note that the error patterns in the same row above have their lowest 1 at the same bit position. For example, in the 4-th row, all the three error patterns have their lowest 1 at the bit position $4k = 16$.

If we denote the number of error patterns with lowest 1 at bit position $ik$ as $x_{m-i}$, we have $x_1 = x_2 = 1, x_3 = 2, x_4 = 3, x_5 = 5, x_6 = 8$ for this example. As we showed in Sections 5.1, the lowest bit position in an error pattern where a 1 can occur is $(t+1)k$. Therefore, the maximal index of the sequence $x_i$ is $m-t-1 = 6$. Based on the numerical values of $x_i$’s, we find that $x_i$’s satisfy the following pattern:

$$x_i = 1,$$

for all $1 \leq i \leq 2$, and

$$x_i = x_{i-2} + x_{i-3} + \cdots + x_1 + 1,$$

for all $2 < i \leq 6$. Indeed, the above two equations make sense. For any $1 \leq i \leq t + 1 = 2$, if an error pattern has its lowest 1 at bit position $(m-i)k$, then by Theorem 2, it cannot have any 1s on the left of that bit position. Therefore, the number of error patterns with the lowest 1 at bit position $(m-i)k$ is just 1, and hence Eq. (24) holds. For any $2 < i \leq 6$, the set of error patterns with the lowest 1 at bit position $(m-i)k$ can be partitioned into $(i-1)$ subsets. The subset
When \( x_i > t \) the total number of error patterns is \( \sum_{j=1}^{m} x_{i-j} + 1 \). Thus, Eq. (25) also holds.

Indeed, in the general case, no matter whether \( l \) is a multiple of \( k \) or not, we have

\[
x_i = \begin{cases} 1, & \text{for all } 1 \leq i \leq t + 1, \\ x_i = x_{i-t-1} + x_{i-t-2} + \cdots + x_1, & \text{for all } t + 1 < i \leq m - t - 1. 
\end{cases}
\]

for all \( t + 1 < i \leq m - t - 1 \). Note that the total number of non-zero error patterns is given by \( x_{m-t-1} + x_{m-t-2} + \cdots + x_1 \). To count the total number of error patterns, we should also include the zero error pattern. Therefore, the total number of error patterns is

\[
N = x_{m-t-1} + x_{m-t-2} + \cdots + x_1 + 1.
\]

If we extend the recursive definition of \( x_i \) shown in Eq. (28) to \( i > m - t - 1 \), then the number of error patterns is just \( x_m \).

Note that by Eq. (27), we also have

\[
x_i = x_{i-t-1} + x_{i-t},
\]

for all \( i > t + 1 \).

In summary, the number of error patterns is given by \( x_m \), where \( x_i \) (\( i \geq 1 \)) is recursively defined by Eq. (26) and (28). When \( t = 0 \), we have \( x_m = 2^{m-1} \). Therefore, the runtime of the proposed algorithm is \( O(2^m) \). When \( t = 1 \), then the sequence \( x_i \) is a Fibonacci sequence and \( x_m \) is the \( m \)-th value in the sequence, which is equal to

\[
\frac{1}{\sqrt{5}} \left( \frac{1 + \sqrt{5}}{2} \right)^m - \frac{1}{\sqrt{5}} \left( \frac{1 - \sqrt{5}}{2} \right)^m.
\]

Therefore, the runtime of the proposed algorithm is \( O \left( \left( \frac{\sqrt{5}+1}{2} \right)^m \right) \).

6 Experimental Results

We implemented our methods for calculating error rate and error distribution using C++. The error distributions of several block-based approximate adders obtained by our method are shown in Section 6.1. Analysis was made to explain the special pattern of the error distributions. We compared the runtime and accuracy of the proposed method to other existing methods in Sections 6.2 and 6.3, respectively.

6.1 Error Distribution Study

Exact error distributions were generated for four block-based approximate adders using our method, which are ETA-IV with \( k = 4 \) and \( l = 2 \), ETA-II with \( k = 4 \) and \( l = 4 \), CSAA with \( k = 4 \) and \( l = 8 \), and a block-based approximate adder with \( k = 4 \) and \( l = 10 \). All these adders are 64-bit in size. The scatter diagrams of the error distributions of the four adders are shown in Fig. 10.

In Fig. 10, each circle represents an error distance and its associated probability. In each plot, the x-axis is the error distance in the \( \log_2 \) scale and the y-axis is the probability also in the \( \log_2 \) scale. We can see that in each figure, the scatter points approximately form a triangle. Inside the triangle, points are located in a regular way. However, if we zoom in the figures, we will find that some circles overlap with others. The circles with a thicker outline or the solid rounded rectangles in the figures are actually resulted by the clustering of a number of circles. The reason for the clustering along the x-axis is that the error distance of an error pattern is dominated by the highest 1 in the error pattern. For example, if the highest 1 of an error pattern appears at the bit position 36, then the 1s on the right of bit position 36 in the error pattern can only appear at bit position lower than or equal to 32. Thus, the error distance is close to \( 2^{36} \) (indeed, the error distance is larger than or equal to \( 2^{36} \)), which are approximately 36 in the \( \log_2 \) scale. Thus, all error patterns which have their leading 1 at the same location are located very closely in the x dimension. Since the leading 1 in an error pattern can only occur at bit positions \( ik \), where \( k = 4 \) and \( i \) is an integer, the clusters are located near \( x = 4i \) in the \( \log_2 \) scale, as shown in the figures.

Moreover, we can also observe from the figures that for points with close error distances, they cluster into a number of groups along the y-axis. The reason behind this is that among the error patterns with the leading 1 at the same location, the patterns with the same number of 1s have similar probabilities. We use the 64-bit CSAA with \( k = 4 \) and \( l = 8 \) to illustrate this. Consider all error patterns with their error distances around 36 in the \( \log_2 \) scale. Their leading 1s are all at bit position 36. Since \( l = 1/k = 2 \), two adjacent 1s in any error pattern must be at least 12 bits (or 3 blocks) away and the lowest bit position that can have a 1 is 12. Thus, there are at most three 1s in these error patterns. We divide them into three sets based on the number of 1s in the error pattern. The first set consists of the error patterns with a single 1. Using the vector representation shown in Fig. 9, the set includes only one error pattern of the form \( [9 \ldots 9 \ldots 9] \). By Theorem 2, its probability is \( d_{9,0} \cdot e_{9,0} \). The second set is composed of error patterns with two 1s. They are \( [9,6,6,6,0], [9,5,5,5,5,0], [9,4,4,4,4,0], [9,3,3,3,3,0] \), respectively. The third set consists of error patterns with three 1s. The set only has one element, which is \( [9,6,6,6,6,6,0] \). Its probability is \( d_{9,6,6,6,6,6,0} \cdot e_{9,6,6,6,6,6,0} \). It can be seen that the probability of an error pattern with \( q \) 1s has \( q e_{i,j} \) terms in its product expression.

Now consider \( e_{i,j} \). Since for the example approximate adder we consider, its \( l \) is a multiple of \( k \), the value \( e_{i,j} \) is given by Eq. (23), i.e., \( e_{i,j} = P(\bar{P}_{i-1}) \cdots P(\bar{P}_{i-j})P(G_{i-1})d_{i-j-1} \). Under the assumption that the inputs are uniformly distributed, the value \( P(\bar{P}_{i-1}) \cdots P(\bar{P}_{i-j})P(G_{i-1}) \) is a constant for a fixed \( t \). We denote this constant as \( c \). Then, we can represent the value of \( e_{i,j} \) as \( c \cdot d_{i-j-1} \). Therefore, the value of \( e_{i,j} \) is proportional to \( d_{i-j-1} \). The sequence of black dots in Fig. 11 shows the values of \( d_i \) for \( 0 \leq i \leq m - 1 \) for the CSAA with \( k = 4 \) and \( l = 8 \). From the figure, we can see that all \( d_i \)’s are very close. As a result, all \( e_{i,j} \)’s for \( 0 \leq i, j \leq m - 1 \) and \( i - j > t \) are also very close. Consequently, we can see that the error patterns in the same set have very close probabilities, since their probability expressions contain the same number of \( e_{i,j} \) terms. Therefore, the error patterns in the same set also cluster in the y dimension. Since there are three sets, the number of clusters is three, as shown in Fig. 10c. Furthermore, since the number of \( e_{i,j} \) terms in the probability expression of an error pattern is equal to the
number of 1s in the error pattern, the probability value of an error pattern in the log₂ scale is proportional to the number of 1s in the error pattern. This explains why the distances along the y dimension between any two adjacent clusters are the same. Similar analysis can be made for points with other error distances.

However, we found that the clustering of the points along the y-axis is not so strong for ETA-IV, as shown in Fig. 10a. The reason lies in that $d_i$'s for $i = 0, 1, \ldots, m-1$ of this adder are quite different from each other, as shown by the sequence of red dots in Fig. 11 making the values of $e_{i,j}$ also vary greatly for different $(t-j) (i-j > t)$. Hence, for error patterns with close error distances, although some of them have the same number of 1s, their probabilities are not very close.

Based on the above analysis, we can see that the number of clusters over the error patterns with close error distance (i.e., error patterns with the leading 1 at the same bit position) is equal to the maximal number of 1s in such error patterns. Suppose the leading 1 is at bit position $i_k$. By Theorem 2, the maximal number of 1s in such error patterns is $\lfloor i/(t+1) \rfloor$. Therefore, the number of clusters on the error patterns with error distance close to $i_k$ (in the log₂ scale) is equal to $\lfloor i/(t+1) \rfloor$. This explains the trend of how the number of clusters increases with the error distance $i_k$ as shown in the figures. For example, for the ETA-II, as shown in Fig. 10b starting from the first error distance $4 \cdot 2$, two adjacent error distances $4i$ and $4(i+1)$ have the same number of clusters. Then, the next two adjacent error distances $4(i+2)$ and $4(i+3)$ have one more cluster than the previous two. This pattern continues until the last error distance. This is because for ETA-II, $t = 1$ and hence the number of clusters for an error distance $i_k$ is equal to $\lfloor i/2 \rfloor$. From the figures, we can also see that the minimum non-zero error distances for the four approximate adders are different, while the maximal error distances are close. The minimal non-zero error distances for ETA-IV, ETA-II, CSAA, and the block-based approximate adder with $k = 4$ and $l = 10$ are 4, 8, 12, and 12 in the log₂ scale, respectively, while the maximal error distances are all close to 60. This is reasonable, since by Theorem 2, for a non-zero error pattern, the lowest and the highest bit positions where a 1 can occur are $(t+1)k$ and $(m-1)k$, respectively.

Given that the error patterns are clustered around error distance $i_k$ in the log₂ scale, we are also interested in studying how the sum of the probabilities of all the error patterns with distance around $i_k$ changes with the value $i_k$. For the four approximate adders above, we obtained the sums and plotted the bar graphs shown in Fig. 11. It can be easily seen that the height of each bar located at error distance $i_k$ (in the log₂ scale) also represents the probability of an error pattern with the leading 1 at the bit position $i_k$. 

---

**Fig. 10:** Scatter diagrams for error distribution produced by our method for: (a) ETA-IV with $k = 4$ and $l = 2$; (b) ETA-II with $k = 4$ and $l = 4$; (c) CSAA with $k = 4$ and $l = 8$; (d) a block-based approximate adder with $k = 4$ and $l = 10$.

**Fig. 11:** $d$ values for ETA-IV with $k = 4$ and $l = 2$, ETA-II with $k = 4$ and $l = 4$, CSAA with $k = 4$ and $l = 8$, and a block-based approximate adder with $k = 4$ and $l = 10$. 
or the probability of an error pattern with error distance in the range $[p^ik, 2^{i+1}k)$. We denote such a probability as $P(LO = ik)$. From the figures, we can see that as $i$ increases, the probability $P(LO = ik)$ also increases or keeps unchanged. However, for different adders, the trends are different. To understand the reason, we will analyze the probability $P(LO = ik)$. Indeed, we have

$$P(LO = ik) = P(e > ik) = 0|e[ik] = 1) P(e[ik] = 1),$$

where $P(e > ik) = 0|e[ik] = 1)$ is the probability that there is no 1 on the left of the bit position $ik$ under the condition that there is a 1 at bit position $ik$, and $P(e[ik] = 1)$ is the probability that there is a 1 at bit position $ik$. From Section 5, we know that $P(e > ik) = 0|e[ik] = 1)$ is $d_{m-i-1}$. Now we only need to obtain $P(e[ik] = 1)$.

If $l$ is a multiple of $k$, then by Theorem 1, we have

$$P(e[ik] = 1) = P(P_{i-1}) \cdots P(P_{i-l})P(G_{i-t}) \text{ and } P(P_{i-1}) \cdots P(P_{i})P(G_{i}) \text{ and } P(P_{i-1}) \cdots P(P_{i-l})P(G_{i-t-1}) \text{ and } P(P_{i-1}) \cdots P(P_{i-1})P(G_{i-t-2}) \cdots .$$

Under the assumption that the inputs are uniformly distributed, the product is a constant for a fixed $t$ and hence, $P(LO = ik)$ is proportional to $d_{m-i-1}$. This can be verified by Fig. 12a and 12c, which show the bar graphs for the ETA-II and CSAA, respectively. Indeed, the trend of how the height of the bar changes with error distance $ik$ follows the same trend of how the sequence $d_{m-t-1}, d_{m-t-2}, \ldots, d_0$ changes, which can be observed from Fig. 11 in which the sequence of blue dots corresponds to the sequence $d_0, d_1, \ldots, d_{m-1}$ for the ETA-II and the sequence of black dots corresponds to the sequence for CSAA. As shown in Fig. 11, for CSAA, the sequence $d_0, d_1, \ldots, d_{m-1}$ decreases very slowly. This explains why the bars in the graph for CSAA increases very slowly.

If $l$ is not a multiple of $k$, then $P(e[ik] = 1)$ can be obtained by Theorem 3. However, in this case, $P(e[ik] = 1)$ for $i = t + 1$ differs from $P(e[ik] = 1)$'s for $i > t + 1$. Indeed, we have

$$P(e[ik] = 1) = \begin{cases} A(i), & i = t + 1 \\ A(i) + B(i), & i > t + 1 \end{cases},$$

where $A(i) = P(P_{i-1}) \cdots P(P_{i-l})P(P_{i-l-1})P(G_{i-t})$ and $B(i) = P(P_{i-1}) \cdots P(P_{i-l})P(P_{i-l-1})P(G_{i-t-2})$. Under the assumption that the inputs are uniformly distributed, both $A(i)$ and $B(i)$ are positive constants independent of $i$. This explains the special pattern shown in Fig. 12d for the block-based approximate adder with $k = 4$ and $l = 10$. From the figure, we can see the heights of the second bar up to the last bar are almost the same. This is because $P(e[ik] = 1)$ are the same for all $i > t + 1$ and the values $d_0, d_1, \ldots, d_{m-1}$ are very close as shown by the sequence of cyan dots in Fig. 11. However, we can see the height of the first bar is obviously shorter than the other bars. This is because $P(e[(t+1)k] = 1)$ is smaller than any other $P(e[ik] = 1)$ ($i > t + 1$) by a positive constant $B(i)$. The above reasoning can be also applied to explain the pattern shown in Fig. 12a for the ETA-IV. Specifically, the trend from the second bar to the last bar follows the trend of the sequence $d_{m-3}, d_{m-4}, \ldots, d_0$, which is shown by the sequence of red dots in Fig. 11. The first bar is short than the second bar because $P(e[(t+2)k] = 1)$ is smaller than $P(e[(t+1)k] = 1)$ and $d_{m-2} < d_{m-3}$.

### 6.2 Runtime Study

In this section, we compared the runtime of our method to obtain error distribution with the Monte Carlo sampling method, which randomly chooses a subset of input combinations for simulation, and the exhaustive method, which enumerates all the input combinations. All the methods were implemented in C++. All the experiments were conducted on a virtual machine running Linux operating system with 1GB memory. The host machine is a 3.1 GHz desktop. We also compared the asymptotic runtime of our method with the exact method proposed in [12].

Fig. 13 shows the runtime of the proposed method to obtain error distribution and the Monte-Carlo methods with 10K, 100K, and 1M samples on four approximate adders: ETA-IV with $k = 4$ and $l = 2$, ETA-II with $k = l = 4$, CSAA with $k = 4$ and $l = 8$, and a block-based approximate adder with $k = 4$ and $l = 10$. For each type of adder, we did experiments on four different block numbers $m = 4, 8, 12, 16$. Thus, the operand sizes $n$ vary from 16 to 64.

From the four plots in Fig. 13, we can see that the proposed method consumes much less time than the Monte-Carlo method. For all the experiments, the proposed method takes less than 0.02 second. For three 64-bit approximate adders with $l = 4$, $l = 8$, and $l = 10$, the proposed method needs less than 0.002 seconds. In contrast, for the Monte-Carlo method with 10K samples, the runtime is 0.05 ~ 0.9 seconds. As the sample size increases, the runtime of the Monte Carlo method also increases linearly. When the sample size reaches 1M, the runtime of the Monte-Carlo method is $5 ~ 90$ seconds.

Comparing the four plots in Fig. 13, we can also see that the runtime of the Monte-Carlo method increases with the carry generator size $l$. This is reasonable since a larger carry generator size $l$ implies longer simulation time. In contrast, the runtime of the proposed method decreases with $l$. Especially, our method for ETA-IV with $l = 2$ takes much more time than the other three adders when the number of blocks are 12 and 16. This is also reasonable. As we discussed in Section 5.3, the runtime of the proposed method is proportional to $N$, where $N$ is the total number of error patterns of a block-based approximate adder. For a fixed $n$ and $k$, as $l$ decreases, the value $t = [l/k]$ decreases. Consequently, the number of error patterns increases. According to our experimental results, a 64-bit ETA-IV with $k = 4$ and $l = 2$ has 32768 error patterns, while a 64-bit ETA-II with $k = 4$ and $l = 4$ has 987 error patterns. The number of error patterns for a 64-bit CSAA with $k = 4$ and $l = 8$ and a 64-bit block-based approximate adder with...
Finally, we also compared the asymptotic runtime of our method with the method proposed in [12]. The asymptotic runtime of the method in [12] is \(O(2^n t)\) and that of our method is \(O(N(m, t))\), where \(N(m, t)\) is the total number of error patterns of a block-based approximate adder, determined by the number of blocks \(m\) and the value \(t = \lceil l/k \rceil\). The value \(N(m, t)\) can be obtained by the method shown in Section 5.3. By ignoring the constant, we used the ratio \(2^m/N(m, t)\) to measure the asymptotic runtime speed-up of our method over the method in [12]. In Fig. 14 for each \(t = 0,1,2,3\), we show the asymptotic runtime speed-up versus \(m\) for \(m\) ranging from 1 to 16. We can see that for block-based approximate adders with \(t = 0\), such as ETA-IV, both methods have the same asymptotic runtime. However, for adders with \(t > 0\), such as ETA-II and CSAA, our method is much more efficient than the method in [12].

### Fig. 14: Asymptotic runtime speed-up of the proposed method over the method in [12].

To show the advantage in accuracy, we used the results of the proposed method as the reference and computed the relative errors of ERs and MSEs obtained by the method in [10] and the Monte Carlo sampling methods with sample sizes as 10K, 100K, and 1M. Note that since the work [10] has given an exact analytical expression for the mean error distance (MED), we did not consider the relative errors of MED in our experiment. The relative errors of ERs and MSEs in percentage are shown in Table 2. The experiments were performed on eight different 32-bit block-based approximate adders. The types of the adders and their \(k\) and \(l\) values are listed in the first column of the table. For each sample size \(M\) in the Monte Carlo simulations, we ran the entire \(M\)-sample simulation 100 times and obtained the relative error for each simulation run. The final relative error listed in the table is the average relative error over 100 runs. The results in the table show that the relative errors on ERs and the MSEs computed by the method in [10] can be up to 4.7% and 8.6%, respectively. For the Monte Carlo method with sample size equal to 10K, the relative errors on ERs and MSEs can be up to 15.0% and 32.8%, respectively. When the sample size increases, the relative errors decrease. However, simulations with a larger sample size takes longer time.

It should be noted that the method in [10] can only estimate the values of ER and MSE. It cannot provide error distributions. However, the Monte-Carlo method could generate an error distribution through random simulation, although the result is not exact. Next, we compared the error distributions given by the Monte-Carlo method to the exact error distributions produced by the proposed method. We applied both methods to generate the type of distribution as shown in Fig. 12. The results on 64-bit ETA-IV with \(k = 4\) and \(l = 2\), ETA-II with \(k = 4\) and \(l = 4\), CSAA with \(k = 4\) and \(l = 8\), and a block-based approximate adder with \(k = 4\) and \(l = 10\) are shown in Fig. 15. The white bars show the exact error distribution produced by our method, while the red lines on each bar indicate the range of probabilities.
in 20 Monte Carlo simulation runs, each with 100K samples. From the figure, we can see that when \( l \) is small, the Monte Carlo simulation can produce a result close to the accurate distribution, but when \( l \) increases, the accuracy of the Monte Carlo simulation degrades. The reason is that when \( l \) increases, the error probability decreases. For example, as shown in Fig. 15, the error probability of ETA-IV is on the scale of 0.01, while the error probability of the block-based approximate adder with \( k = 4 \) and \( l = 10 \) is on the scale of 0.0001. As the probability becomes smaller, the relative variation of the Monte Carlo result becomes larger. This clearly indicates the accuracy of the Monte Carlo method is reduced when the error probability is small, which is not uncommon for approximate adders. In contrast, our method is guaranteed to obtain the exact distribution.

7 Conclusion

In this paper, we proposed an accurate and efficient method to obtain the error rate and error distribution of block-based approximate adders. The method to obtain error distribution indeed achieves the theoretical lower bound on the asymptotic runtime. Once the distribution is known, some other error metrics of interest, such as mean error distance and mean square error, can be easily obtained. Experimental results demonstrated that the proposed methods are accurate and efficient. Compared to the Monte Carlo sampling method, our method is much better, especially when the error probability is small.

In this work, we considered block-based approximate adders with the speculated carry-in to the carry generator as 0. It is also possible to use the input bit at one bit position lower as the speculated carry-in \([18]\). In our future work, we will develop techniques to analyze the error statistics for this type of approximate adders.

| \( \eta \) | ETA-IV | ETA-II | CSAA |
|---|---|---|---|
| \( k = 2, l = 1 \) | \( k = 2, l = 4 \) | \( k = 4, l = 8 \) | \( k = 4, l = 10 \) |
| \( \text{ER} \) | \( \text{ER} \) | \( \text{ER} \) | \( \text{ER} \) |
| \( \text{MSE} \) | \( \text{MSE} \) | \( \text{MSE} \) | \( \text{MSE} \) |
| \( 0.812 \) | \( 4.658 \) | \( 0.256 \) | \( 4.935 \) |
| \( 8.573 \) | \( 1.541 \) | \( 0\) | \( 14.399 \) |
| \( 0.037 \) | \( 0.098 \) | \( 0 \) | \( 4.654 \) |

Fig. 15: Comparison between our method and the Monte Carlo sampling method with 100K samples in producing error distribution for: (a) ETA-IV with \( k = 4 \) and \( l = 2 \); (b) ETA-II with \( k = 4 \) and \( l = 4 \); (c) CSAA with \( k = 4 \) and \( l = 8 \); (d) a block-based approximate adder with \( k = 4 \) and \( l = 10 \).

This work is supported by National Natural Science Foundation of China (NSFC) under Grant No. 61574089 and 61472243.

ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China (NSFC) under Grant No. 61574089 and 61472243.

REFERENCES

[1] J. Han and M. Orshansky, “Approximate computing: An emerging paradigm for energy-efficient design,” in *European Test Symposium*, 2013, pp. 1–6.
[2] H. R. Mahdian, 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*, vol. 57, no. 4, pp. 850–862, 2010.
[3] 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.
[4] A. K. Verma, F. Brisk, and P. Ferenc, “Variable latency speculative addition: A new paradigm for arithmetic circuit design,” in *Design, Automation and Test in Europe*, 2008, pp. 1250–1255.
[5] N. Zhu, W. L. Goh, and K. S. Yeo, “An enhanced low-power high-speed adder for error-tolerant application,” in *International Symposium on Integrated Circuits*, 2009, pp. 69–72.
[6] Y. Kim, Y. Zhang, and P. Li, “An energy efficient approximate adder with carry skip for error resilient neuromorphic vlsi systems,” in *International Conference on Computer-Aided Design*, 2013, pp. 130–137.
[7] A. B. Kahng and S. Kang, “Accuracy-configurable adder for approximate arithmetic designs,” in *Design Automation Conference*, 2012, pp. 820–825.
[8] J. Hu and W. Qian, “A new approximate adder with low relative error and correct sign calculation,” in *Design, Automation and Test in Europe*, 2015, pp. 1449–1454.
[9] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, “On reconfiguration-oriented approximate adder design and its application,” in *International Conference on Computer-Aided Design*, 2013, pp. 48–54.
[10] L. Li and H. Zhou, “On error modeling and analysis of approximate adders,” in *International Conference on Computer-Aided Design*, 2014, pp. 511–518.
[11] C. Liu, J. Han, and F. Lombardi, “An analytical framework for evaluating the error characteristics of approximate adders,” *IEEE Transactions on Computers*, vol. 66, no. 5, pp. 1268–1281, 2015.
[12] S. Mazahir, O. Hasan, R. Hatiz, M. Shafique, and J. Henkel, “Probabilistic error modeling for approximate adders,” *IEEE Transactions on Computers*, vol. PP, no. 99, pp. 1–14, 2016.
[13] K. Du, P. Varman, and K. Mohanram, “High performance reliable variable latency carry select addition,” in *Design, Automation and Test in Europe*, 2012, pp. 1257–1262.
[14] I. C. Lin, Y. M. Yang, and C. C. Lin, “High-performance low-power carry speculative addition with variable latency,” *Very Large Scale Integration Systems IEEE Transactions on*, vol. 23, no. 9, pp. 1591–1603, 2015.
[15] S.-L. Lu, “Speeding up processing with approximation circuits,” *Computer*, vol. 37, no. 3, pp. 67–73, 2004.
[16] N. Zhu, W. L. Goh, G. Wang, and K. S. Yeo, “Enhanced low-power high-speed adder for error-tolerant application,” in *International SoC Design Conference*, 2010, pp. 323–327.
[17] M. Shafique, W. Ahmad, R. Hatiz, and J. Henkel, “A low latency generic accuracy configurable adder,” in *Design Automation Conference*, 2015, pp. 861–866.
[18] J. Huang, J. Lach, and G. Robins, “A methodology for energy-quality tradeoff using imprecise hardware,” in *Design Automation Conference*, 2012, pp. 504–509.