

# An Efficient Method for Calculating the Error Statistics of Block-based Approximate Adders

Yi Wu<sup>†</sup>, You Li<sup>†</sup>, Xiangxuan Ge<sup>†</sup>, Yuan Gao, 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, the other important design metrics are the various error statistics, such as error rate (ER), mean error distance (MED), and mean square error (MSE). Given the popularity of block-based approximate adders, in this work, we propose an efficient method to obtain the error statistics of these adders. We first show how to calculate the error rate. Then, we demonstrate an approach to get the error distribution, which can be used to calculate other error statistics, such as MED and MSE. **Our method is applicable to an arbitrary block-based approximate adder. It is accurate for the uniformly distributed inputs. Experimental results also demonstrated that it produces error metrics close to the accurate ones for various types of non-uniform input distributions. Compared to the state-of-the-art algorithm in analyzing error statistics of block-based approximate adders, our method significantly improves the runtime with the same accuracy.**

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

## 1 INTRODUCTION

REDUCING power consumption is becoming increasingly significant in today's VLSI design. Meanwhile, many applications that exhibit an inherent tolerance to errors in computation, such as multimedia, signal processing, and data mining, become more widely used recently with the prevalence of mobile and embedded computing systems. **Given this context, approximate computing, a novel computing paradigm that leverages the inherent resilience of applications to trade quality for power, has been proposed [1], [2]. The research topics on approximate computing are diverse, including approximate circuit design [3], quality modeling of approximate hardware [4], approximate logic synthesis [5], etc. The effectiveness of approximate computing has been demonstrated at various design levels ranging from algorithms [6], architectures [7], down to logic [8], and layout [9].**

Arithmetic circuits are critical components to many error-tolerant applications. As a result, researchers have been focusing on designing various kinds of approximate arithmetic units. A large number of previous works focused on approximate adders [10], [11], [12], [13], [14], [15], [16], [17], [18] and multipliers [19], [20], [21], [22], which are the most fundamental arithmetic circuits. **Additionally, approximate hardware accelerators also attracted a lot of research efforts recently [23], [24], [25], [26].**

Among the many arithmetic operations, addition is the most important one, since it is vital in supporting more complex arithmetic operations, such as multiplication and division. Therefore, we focus on approximate adders in this work. Generally speaking, there are two design types for approximate adders in the literature. The first type replaces the 1-bit full adders at less significant bit positions by a simpler but inaccurate module. For example, to substitute the accurate 1-bit full adder at the lower bit positions,

an OR gate is used in the Low-Part-OR adder [10] and an approximate mirror adder is used in [11]. The most significant bits are 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* [27]. It divides the entire adder into a number of blocks, each containing a sub-adder. The calculation of the sum in each block exploits the carry speculation mechanism. It is based on the observation that a 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 is reduced.** Furthermore, this type of adder generally has a low error rate. Many available approximate adders fall into this category. Examples include Almost Correct Adder (ACA-I) [12], Error Tolerant Adder Type II (ETA-II) [13], Carry-Skip Approximate Adder (CSAA) [14], **Gracefully-Degrading Adder (GDA) [17], and Generic Accuracy Configurable Adder (GeAr) [18]** (More details of these approximate adders will be discussed in Section 2). Given its popularity and better accuracy, we focus on block-based approximate adder in our work.

To measure the quality of an approximate adder, besides the normal metrics such as area, delay, and power consumption, we also need error statistics, including error rate (ER), mean error distance (MED), and mean square error (MSE). 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 ER was studied, but neither MED nor MSE was reported [12] [28].

To address the above problems, several recent works [27], [29], [30], [31] have proposed methods for obtaining important error metrics of approximate adders. **In [29], a recursive method was presented to compute the ER of low power approximate adders, which are cascade of approximate 1-bit full adders. The targeted approximate adders belong to the first type we mention above, which are**

- Yi Wu, You Li, Xiangxuan Ge, Yuan Gao, and Weikang Qian are with the University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, Shanghai, China, 200240.
- E-mail: {eejessie, you.li, gxx, plateau, qianwk}@sjtu.edu.cn
- <sup>†</sup>These authors contributed equally.

different from block-based approximate adders. The other three works (i.e., [27], [30], [31]) focus on the block-based approximate adders. The work [30] proposed an analytical framework to evaluate the error statistics of three types of adders: ACA-I, Equal Segmentation Adder (ESA), and ETA-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 [27], which can be applied to a wider range of approximate adders. It gives an accurate analysis on MED, but ER and MSE are still estimates. Also, none of [27] and [30] showed how to obtain the error distributions, **which is the most fundamental representation for the computation error and can be used to derive many error metrics of interest**. A recent work [31] proposed a generic analytical method to obtain the ERs and error distributions for block-based approximate adders. This method is accurate for the uniform input distribution and approximate for non-uniform input distributions. It is essentially an enumeration method, where the conditions on inputs that can lead to errors in each block are identified. Since these conditions are not mutually exclusive, this method spends much effort in identifying and eliminating overlaps among those conditions. Thus, it has long runtime.

In this work, we propose a novel efficient method to obtain the ER and error distribution of general block-based approximate adders. With the error distribution known, many other error metrics, such as MED and MSE, can be easily derived. **Our method is exact for the uniform input distribution**. For various types of non-uniform input distributions, our method can generate results close to the exact ones, as demonstrated by the experimental results. Compared to the previous analytical approaches [27] and [30], our method is able to give the exact error distribution for general block-based approximate adders **under the uniform input distribution**. Compared to the previous work [31], our method avoids redundant computation and hence, is much faster. Indeed, its asymptotic runtime<sup>1</sup> for generating the error distribution reaches the theoretical lower bound. The proposed method provides an important aid to designers in choosing a proper approximate adder.

The remainder of the paper is organized as follows. Section 2 introduces the general model of a block-based approximate adder and links it to some previously proposed approximate adders. Section 3 discusses some preliminaries. Section 4 presents our method for calculating the ER. Sections 5 and 6 describe the theory and algorithm for obtaining the error distribution, respectively. Section 7 shows for what type of input distributions the proposed methods give the correct ER and error distribution. Section 8 presents the experimental results. Finally, Section 9 concludes the paper.

## 2 BLOCK-BASED APPROXIMATE ADDERS

A representative of block-based adder is Error Tolerant Adder Type II (ETA-II) shown in Fig. 1. It divides the entire  $n$ -bit adder into  $m$  sub-adders of equal bit length of  $k = n/m$  [13]. 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.

A general model of the block-based approximate adder [27], [31] is shown in Fig. 2. In the model, the number of bits is  $n$ . Assume the two inputs of an  $n$ -bit adder are

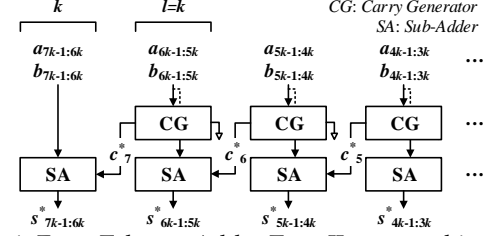


Fig. 1: Error Tolerant Adder Type II proposed in [13].

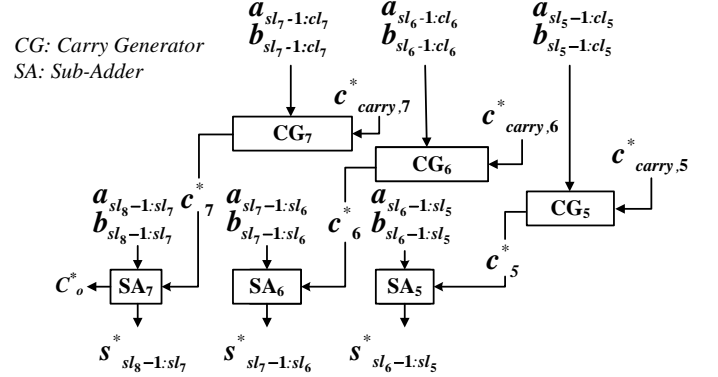


Fig. 2: General model of a block-based approximate adder.

$A = a_{n-1} \dots a_0$  and  $B = b_{n-1} \dots b_0$ . The approximate sum is denoted by  $S^* = s_{n-1}^* \dots s_0^*$ . The carry-out of the entire approximate adder is denoted by  $c_o^*$ . The sum in this model is divided into multiple blocks that are calculated separately. The total number of blocks is denoted by  $m$ . **The block of the least significant bits (i.e., the rightmost block) has the index of 0 and the block index increases from right to left.**

In general cases, the lengths of all the blocks in an approximate adder can be unequal. We denote the length of block  $i$  ( $0 \leq i \leq m-1$ ) as  $k_i$ . For simplicity, we use  $sl_i$  to represent the lowest bit position of block  $i$ . By the definitions, we clearly have  $sl_0 = 0$  and for  $1 \leq i \leq m-1$ ,  $sl_i = \sum_{j=0}^{i-1} k_j$ . The sum bits in block  $i$  ( $0 \leq i \leq m-1$ ) of the approximate adder are  $s_{sl_{i+1}-1}^* \dots s_{sl_i}^*$ . They are generated by a sub-adder taking a speculated carry-in  $c_i^*$ . **In the case of accurate adder**, the carry-in should be produced by **all** the input bits lower than position  $sl_i$ . However, for the block-based approximate adder,  $c_i^*$  is produced by a truncated carry generator of length  $l_i$ , as shown in Fig. 2. The lengths  $l_i$ 's can also be different for different  $i$ 's. We denote the truncated carry generator for block  $i$  as  $CG_i$ . The inputs of  $CG_i$  are from bit position  $(sl_i - 1)$  to bit position  $(sl_i - l_i)$ . For simplicity, we define  $cl_i = sl_i - l_i$ , which denotes the lowest bit position of carry generator  $CG_i$ . Carry generator  $CG_i$  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 sub-adder 0, it has no carry generator and its speculated carry-in  $c_0^*$  is always equal to the correct carry-in of value 0. Thus, for sub-adder 0, we have  $l_0 = 0$  and  $cl_0 = sl_0 - l_0 = 0$ . The carry-out  $c_o^*$  of the entire adder is produced by the leftmost sub-adder.**

In summary, a block-based approximate adder is characterized by the adder size  $n$ , the number of blocks  $m$ , the length of block  $i$ ,  $k_i$ , for  $0 \leq i < m$ , and the length of the carry generator of block  $i$ ,  $l_i$ , for  $1 \leq i < m$ . Many previously proposed approximate adders fit into this

1. Asymptotic runtime is the time complexity of an algorithm commonly expressed using asymptotic notation. It reflects how the running time of an algorithm increases with the size of the input in the limit [32].

model. For example, Almost Correct Adder (ACA-I) [12], Speculative Carry Select Adder (SCSA) [28], ETA-II, Error Tolerant Adder Type IV (ETA-IV) [33], Carry-Skip Approximate Adder (CSAA) [14], and Gracefully-Degrading Adder (GDA) [17] all have equal block lengths, i.e., they have  $k_{m-1} = \dots = k_1 = k_0 = k$ , where  $k$  is a constant. However, they differ in the values of  $k_i$ 's and the relationships between  $k_i$ 's and  $l_i$ 's. For example, ACA-I corresponds to the model where  $k_i = 1$  for all  $0 \leq i < m$ . ETA-II and SCSA correspond to the model where  $l_i = k_i = k$  for all  $1 \leq i < m$ . ETA-IV corresponds to the model where  $l_i = k_i/2 = k/2$  for all  $1 \leq i < m$ . CSAA corresponds to the model where  $l_i = 2k_i = 2k$  for all  $2 \leq i < m$  and  $l_1 = k_1 = k$ . GDA corresponds a model where the  $l_i$ 's are not necessarily equal and hence, it is more flexible. Some other block-based approximate adders have unequal block lengths. For example, Accuracy-Configurable Adder (ACA-II) [15] corresponds to the model where  $k_0 = 2k$  and  $k_i = l_i = k$  ( $1 \leq i < m$ ), where  $k$  is a constant. Generic Accuracy Configurable Adder (GeAr) [18] corresponds to the model where  $k_{m-1} = \dots = k_1 = k$ ,  $l_{m-1} = \dots = l_1 = l$ , and  $k_0 = l + k$ , where  $k$  and  $l$  are two constants that are not necessarily equal. Error Tolerant Adder II-Modified (ETA-IIM) [13] corresponds to the model where  $k_i$ 's ( $0 \leq i < m$ ) and  $l_i$ 's ( $1 \leq i < m$ ) can be any multiple of a constant  $k$ .

Another common characteristic of these available block-based approximate adders is that for any  $0 < i < m$ , we have  $cl_i \geq cl_{i-1}$ . If this condition is not satisfied, then there exists an  $0 < i < m$  such that  $cl_i < cl_{i-1}$ . This means that the carry generator of a higher block covers that of a lower block. In this case, by reusing part of the signals of the carry generator  $CG_i$ , we can increase the length of the carry generator  $CG_{i-1}$  so that  $cl_{i-1} = cl_i$ . This change will not affect the delay of the adder, since the delay of  $CG_i$  is not changed and it is still larger than the delay of  $CG_{i-1}$ . Meanwhile, by making the length of  $CG_{i-1}$  longer, the accuracy of the entire adder could be improved. In summary, a reasonable block-based approximate adder should not have an  $0 < i < m$  such that  $cl_i < cl_{i-1}$ . Therefore, throughout this paper, we assume that for any block  $i$  ( $0 < i < m$ ), we have  $cl_i \geq cl_{i-1}$ .

### 3 PRELIMINARIES

In this section, we show some preliminaries that will be used in our later analysis.

#### 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

$$p_i = a_i \oplus b_i, g_i = a_i \cdot b_i, k_i = \bar{a}_i \cdot \bar{b}_i.$$

For a group of bits from position  $i_1$  to  $i_2$  ( $i_1 \geq i_2$ ), we denote its group propagate, generate, and kill signals as  $P_{i_1:i_2}$ ,  $G_{i_1:i_2}$ , and  $K_{i_1:i_2}$ , respectively. Their values can be calculated as follows:

$$\begin{aligned} P_{i_1:i_2} &= \prod_{j=i_2}^{i_1} p_j, \\ G_{i_1:i_2} &= \sum_{j=i_2}^{i_1} g_j \prod_{d=j+1}^{i_1} p_d, \\ K_{i_1:i_2} &= \sum_{j=i_2}^{i_1} k_j \prod_{d=j+1}^{i_1} p_d. \end{aligned}$$

By the above definitions,  $P_{i_1:i_2} = 1$  if and only if each pair of input bits from position  $i_1$  to  $i_2$  ( $i_1 \geq i_2$ ) propagates the carry.  $G_{i_1:i_2} = 1$  ( $K_{i_1:i_2} = 1$ ) if and only if among all the bit positions from  $i_1$  to  $i_2$  ( $i_1 \geq i_2$ ), there exists a pair

of input bits that generates (kills) a carry and all the pairs of input bits on the left of that pair propagate the carry. If  $G_{i_1:i_2} = 1$  ( $K_{i_1:i_2} = 1$ ), the carry-out based on the input bits from position  $i_1$  to  $i_2$  will always be the correct value of 1 (0) no matter its carry-in is correct or not. Only when  $P_{i_1:i_2} = 1$  does the carry-out depend on the carry-in signal, which could be wrong. For each combination of input bits from position  $i_1$  to  $i_2$  ( $i_1 \geq i_2$ ), it satisfies exactly one of the three equations:  $P_{i_1:i_2} = 1$ ,  $G_{i_1:i_2} = 1$ , and  $K_{i_1:i_2} = 1$ . This is an important property we will exploit in deriving our efficient method for calculating ER and error distribution. Note that in cases where  $i_1 < i_2$ , we define  $P_{i_1:i_2} = 1$ ,  $G_{i_1:i_2} = K_{i_1:i_2} = 0$ .

We denote the probabilities of  $P_{i_1:i_2}$ ,  $G_{i_1:i_2}$ , and  $K_{i_1:i_2}$  being one as  $P(P_{i_1:i_2})$ ,  $P(G_{i_1:i_2})$ , and  $P(K_{i_1:i_2})$ , respectively. If the two inputs are independent and uniformly distributed, these probabilities are

$$P(P_{i_1:i_2}) = \frac{1}{2^{i_1-i_2+1}}, P(G_{i_1:i_2}) = P(K_{i_1:i_2}) = \frac{1}{2} - \frac{1}{2^{i_1-i_2+2}}.$$

If the inputs are not uniformly distributed, these probabilities can be calculated by a method presented in [31].

In the special case where  $i_1 = sl_{i+1} - 1$  and  $i_2 = sl_i$  for an arbitrary  $i$ , the bits from position  $i_1$  to  $i_2$  constitute all the bits of block  $i$ . For simplicity, we just use  $P_i$ ,  $G_i$ , and  $K_i$  to denote the group propagate, generate, and kill signals of the input bits in block  $i$  ( $0 \leq i < m$ ), respectively. We use  $P(P_i)$ ,  $P(G_i)$ , and  $P(K_i)$  to denote their probabilities of being one, respectively.

#### 3.2 Typical Error Metrics

Typical error metrics of an approximate arithmetic circuit include error rate (ER), mean error distance (MED), and mean square error (MSE) [27]. To introduce them, we first introduce error distance (ED). It is defined as the difference of the approximate sum  $S^*$  and the accurate sum  $S$ , i.e.,  $ED = |S^* - S|$ .

ER is defined as the percentage of input combinations for which the approximate adder produces a wrong result, i.e., a non-zero ED. Mathematically, it is calculated as

$$ER = P(ED \neq 0).$$

MED is the mean value of all the EDs. MSE is the mean value of the squares of all the EDs. Mathematically, they are calculated as

$$\begin{aligned} MED &= E[ED] = \sum_{ED_i \in \Omega} ED_i P(ED_i), \\ MSE &= E[ED^2] = \sum_{ED_i \in \Omega} ED_i^2 P(ED_i), \end{aligned}$$

where  $\Omega$  is the set of all EDs.

### 4 CALCULATING THE ERROR RATE

In this section, we introduce our method for calculating the ER. Note that throughout this section and Sections 5 and 6, we assume that each input is bit-wise independent. This assumption is valid when two inputs are independent and uniformly distributed. Under this assumption, our method can give the exact ER. Further, the technique developed for ER computation can be utilized to obtain the exact error distribution, as will be shown in Section 5.

As can be seen in Fig. 2, 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 ER, we

define  $D_i$  as the event in which all the speculated carry-ins  $c_i^*, c_{i-1}^*, \dots, c_0^*$  are correct. We denote the probability for event  $D_i$  to occur as  $d_i$ . Thus, the ER equals  $1 - d_{m-1}$ . In the following, we will derive a recursive formula to calculate  $d_i$ . We denote the correct carry-in to sub-adder  $i$  as  $c_i$ .

By the approximate adder model, we have  $d_0 = 1$ . Furthermore, we have the assumption that  $0 \leq cl_1 \leq cl_2 \leq \dots \leq cl_{m-1}$ , where  $cl_i$  is the lowest bit position of carry generator  $CG_i$ . If there exists a  $1 \leq t \leq m-1$  such that  $0 = cl_t < cl_{t+1}$ , then  $cl_1 = \dots = cl_t = 0$ . This indicates that for any  $1 \leq i \leq t$ , carry generator  $CG_i$  covers all the lower input bits. Therefore,  $c_i^*, \dots, c_1^*$  are all correct. Consequently,  $d_t = \dots = d_1 = 1$ . What we are interested in is the  $d_i$  values for these  $i$ 's such that carry generator  $CG_i$  does not cover all the lower input bits. Next, we show how we calculate these  $d_i$ 's.

For each block  $i$ , we use  $t_i \geq 0$  to denote the number of blocks that are completely covered by carry generator  $CG_i$ . Mathematically,  $t_i$  is the number satisfying that

$$\sum_{j=i-t_i}^{i-1} k_j \leq l_i < \sum_{j=i-t_i-1}^{i-1} k_j,$$

where  $l_i$  is the length of  $CG_i$ . We define

$$k'_i = l_i - \sum_{j=i-t_i}^{i-1} k_j.$$

Thus, carry generator  $CG_i$  completely covers blocks  $(i-1), \dots, (i-t_i)$  plus the left  $k'_i$  bits of block  $(i-t_i-1)$ .

Note that  $0 \leq k'_i < k_{i-t_i-1}$ . When  $k'_i = 0$ , carry generator  $CG_i$  does not include any bits in block  $(i-t_i-1)$ . When  $k'_i > 0$ , we divide block  $(i-t_i-1)$  into the left group of  $k'_i$  bits and the right group of  $(k_{i-t_i-1} - k'_i)$  bits. For each block  $i$ , the left group of bits in block  $(i-t_i-1)$  that is covered by  $CG_i$  has its group propagate and generate signals as  $P_{j_1:j_2}$  and  $G_{j_1:j_2}$ , respectively, where  $j_1 = sl_{i-t_i} - 1$  and  $j_2 = sl_{i-t_i} - k'_i$ . For simplicity, we denote them as  $PL_i$  and  $GL_i$ , respectively. For the right group of bits in block  $(i-t_i-1)$  that is not covered by  $CG_i$ , its group propagate and generate signals are  $P_{h_1:h_2}$  and  $G_{h_1:h_2}$ , respectively, where  $h_1 = sl_{i-t_i} - k'_i - 1$  and  $h_2 = sl_{i-t_i} - 1$ . For simplicity, we denote them as  $PR_i$  and  $GR_i$ , respectively. The probabilities of these signals being one are denoted as  $P(PL_i)$ ,  $P(GL_i)$ ,  $P(PR_i)$ , and  $P(GR_i)$ , respectively.

The basic idea to obtain the probability  $d_i$  is to identify all input cases that could cause event  $D_i$  to happen. To achieve this, we examine the input bits block by block from block  $(i-1)$  to block 0 and decide for each block  $j$  ( $0 \leq j \leq i-1$ ), which of the three input cases of  $P_j = 1$ ,  $G_j = 1$ , and  $K_j = 1$  could make  $D_i$  happen. Then, we calculate the probabilities of all input cases that make  $D_i$  happen. The probability  $d_i$  is just the sum of all these individual probabilities.

Next, we illustrate how our method obtains  $d_i$  for a fixed  $i$  using an example in which  $t_i = 2$  and  $k'_i > 0$ . The condition that  $t_i = 2$  and  $k'_i > 0$  indicates that the carry generator of block  $i$  covers the entire blocks  $(i-1)$  and  $(i-2)$  plus the left  $k'_i$  bits of block  $(i-3)$ .

First, consider the input bits at block  $(i-1)$ . They satisfy either  $G_{i-1} = 1$ ,  $K_{i-1} = 1$ , or  $P_{i-1} = 1$ . If the bits satisfy that  $G_{i-1} = 1$ , as shown in Fig. 3(a), the speculated carry-in  $c_i^*$  is equal to 1, which is same as the correct carry-in  $c_i$ . Thus, event  $D_i$  happens if and only if  $c_{i-1}^*, \dots, c_0^*$  are

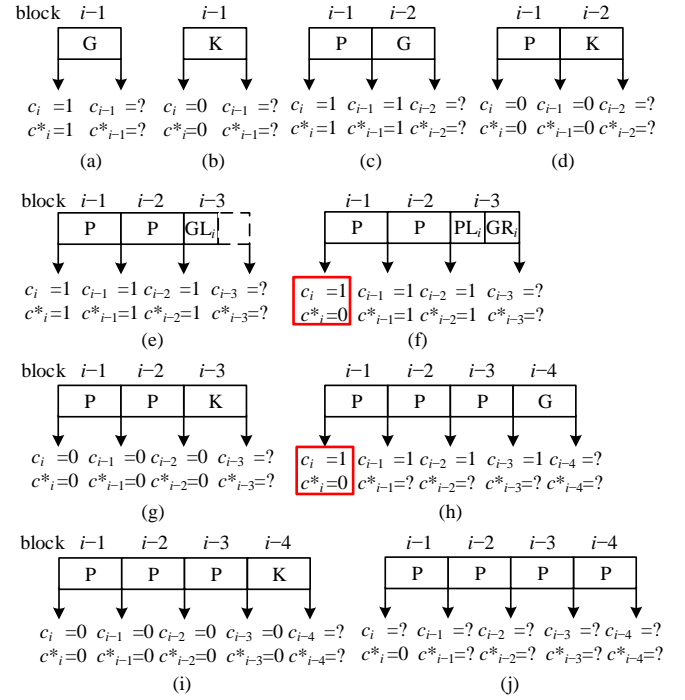


Fig. 3: The speculated and correct carry-ins of blocks for some input cases for the example where  $t_i = 2$  and  $0 < k'_i < k_{i-3}$ . (a)  $G_{i-1} = 1$ ; (b)  $K_{i-1} = 1$ ; (c)  $P_{i-1} = G_{i-2} = 1$ ; (d)  $P_{i-1} = K_{i-2} = 1$ ; (e)  $P_{i-1} = P_{i-2} = GL_i = 1$ ; (f)  $P_{i-1} = P_{i-2} = PL_i = GR_i = 1$ ; (g)  $P_{i-1} = P_{i-2} = K_{i-3} = 1$ ; (h)  $P_{i-1} = P_{i-2} = P_{i-3} = G_{i-4} = 1$ ; (i)  $P_{i-1} = P_{i-2} = P_{i-3} = K_{i-4} = 1$ ; (j)  $P_{i-1} = P_{i-2} = P_{i-3} = P_{i-4} = 1$ .

correct, which means the input bits from block  $(i-2)$  to 0 make event  $D_{i-1}$  happen. Thus, we have

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

where the last equation is due to the independence assumption among the input bits.

The same conclusion holds if the input bits at block  $(i-1)$  satisfy that  $K_{i-1} = 1$ , as shown in Fig. 3(b). Thus, we have

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

If the input bits at block  $(i-1)$  satisfy  $P_{i-1} = 1$ , then we further consider the bits at block  $(i-2)$ . We also classify 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. 3(c)) and the case where  $K_{i-2} = 1$  (shown in Fig. 3(d)), the speculated carry-in  $c_i^*$  generated by  $CG_i$  is equal to the correct carry-in  $c_i$ . Since we assume that the lowest bit of carry generator  $CG_i$  is always on the left or at the same position of the lowest bit of carry generator  $CG_{i-1}$  (i.e.,  $cl_i \geq cl_{i-1}$ ),  $CG_{i-1}$  should completely cover the inputs at block  $(i-2)$ . Thus, the speculated carry-in  $c_{i-1}^*$  generated by  $CG_{i-1}$  is also equal to the correct carry-in  $c_{i-1}$ . Thus, event  $D_i$  happens if and only if  $c_{i-2}^*, \dots, c_0^*$  are correct, which means the input bits from block  $(i-3)$  to 0 make event  $D_{i-2}$  happen. Thus, we have

$$P(D_i, P_{i-1} = G_{i-2} = 1) = P(P_{i-1})P(G_{i-2})P(D_{i-2}), \quad (3)$$

$$P(D_i, P_{i-1} = K_{i-2} = 1) = P(P_{i-1})P(K_{i-2})P(D_{i-2}). \quad (4)$$

If the input bits at blocks  $(i-1)$  and  $(i-2)$  satisfy none of the above cases, then we have  $P_{i-1} = P_{i-2} = 1$ . We further



consider the bits at block  $(i - 3)$ . We need to distinguish the following **four** cases:

- 1) The input bits satisfy that  $GL_i = 1$ , as shown in Fig. 3(e). We can show that  $c_i^* = c_i = 1$ ,  $c_{i-1}^* = c_{i-1} = 1$ , and  $c_{i-2}^* = c_{i-2} = 1$ . Thus, event  $D_i$  happens if and only if  $c_{i-3}^*, \dots, c_0^*$  are correct, which means the bits from block  $(i - 4)$  to 0 make event  $D_{i-3}$  happen. Thus, we have

$$\begin{aligned} P(D_i, P_{i-1} = P_{i-2} = GL_i = 1) \\ = P(P_{i-1})P(P_{i-2})P(GL_i)P(D_{i-3}). \end{aligned} \quad (5)$$

- 2) The input bits satisfy that  $PL_i = GR_i = 1$ , as shown in Fig. 3(f). In this case, the correct carry-in  $c_i = 1$ . However, the speculated carry-in  $c_i^*$  produced by  $CG_i$  is 0, since each bit position in  $CG_i$  propagates and the carry-in to  $CG_i$  is 0. Since  $c_i^* \neq c_i$ , event  $D_i$  cannot happen. Thus, we have

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

- 3) The input bits satisfy that  $K_{i-3} = 1$ , as shown in Fig. 3(g). In this case, it can be shown that  $c_j^* = c_j = 0$ , for  $j = i, i - 1, i - 2$ . Thus, event  $D_i$  happens if and only if the bits from block  $(i - 4)$  to 0 make event  $D_{i-3}$  happen. Thus, we have

$$\begin{aligned} P(D_i, P_{i-1} = P_{i-2} = K_{i-3} = 1) \\ = P(P_{i-1})P(P_{i-2})P(K_{i-3})P(D_{i-3}). \end{aligned} \quad (6)$$

- 4) The input bits satisfy that  $P_{i-3} = 1$ . In this case, we need to continue checking the bits 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 input bits at block  $(i - 4)$ . Since the carry generator of block  $i$  does not cover any bit in block  $(i - 4)$ , there is no need to divide block  $(i - 4)$  into the left group and the right group as before. Therefore, we only need to consider three different situations for block  $(i - 4)$ :  $G_{i-4} = 1$ ,  $K_{i-4} = 1$ , and  $P_{i-4} = 1$ . They are shown in Fig. 3(h), (i), and (j), respectively.

Since  $P_{i-1} = P_{i-2} = P_{i-3} = 1$  and  $t_i = 2$ , the speculated carry-in  $c_i^*$  produced by  $CG_i$  is 0. For the case where  $G_{i-4} = 1$ , since the correct carry-in  $c_i = 1 \neq c_i^*$ , event  $D_i$  cannot happen. Therefore, we have

$$P(D_i, P_{i-1} = P_{i-2} = P_{i-3} = G_{i-4} = 1) = 0.$$

For the case where  $K_{i-4} = 1$ , it can be shown that  $c_j^* = c_j = 0$ , for  $j = i, \dots, i - 3$ , as illustrated in Fig. 3(i). Thus, event  $D_i$  happens if and only if the input bits from block  $(i - 5)$  to 0 make the event  $D_{i-4}$  happen. Thus, we have

$$\begin{aligned} P(D_i, P_{i-1} = P_{i-2} = P_{i-3} = K_{i-4} = 1) \\ = P(P_{i-1})P(P_{i-2})P(P_{i-3})P(K_{i-4})P(D_{i-4}). \end{aligned} \quad (7)$$

For the case where  $P_{i-4} = 1$ , we continue analyzing the inputs of the next block (i.e., block  $(i - 5)$ ) in the same way.

By the same reasoning used for the case where  $P_{i-1} = P_{i-2} = P_{i-3} = 1$ , we can obtain that for any  $5 \leq j \leq i$ , if the input bits from block  $(i - 1)$  to block  $(i - j)$  satisfy that  $P_{i-1} = \dots = P_{i-j+1} = G_{i-j} = 1$ , event  $D_i$  cannot happen. If the bits satisfy  $P_{i-1} = \dots = P_{i-j+1} = K_{i-j} = 1$ , event  $D_i$  happens if and only if the bits from block  $(i - j - 1)$  to 0 make event  $D_{i-j}$  happen. The equation to calculate the probability is similar to Eq. (7). It is

$$\begin{aligned} P(D_i, P_{i-1} = \dots = P_{i-j+1} = K_{i-j} = 1) \\ = P(P_{i-1}) \dots P(P_{i-j+1})P(K_{i-j})P(D_{i-j}). \end{aligned} \quad (8)$$

Finally, for the remaining input case where  $P_{i-1} = \dots = P_0 = 1$ , event  $D_i$  happens. Thus, we have

$$P(D_i, P_{i-1} = \dots = P_0 = 1) = P(P_{i-1}) \dots P(P_0). \quad (9)$$

By the above discussion, we can calculate  $d_i$  as the sum of probabilities of all disjoint input cases that could make event  $D_i$  happen. The expression for  $d_i$  is

$$\begin{aligned} d_i = P(D_i) = & P(G_{i-1})P(D_{i-1}) + P(K_{i-1})P(D_{i-1}) \\ & + P(P_{i-1})P(G_{i-2})P(D_{i-2}) + P(P_{i-1})P(K_{i-2})P(D_{i-2}) \\ & + P(P_{i-1})P(P_{i-2})P(GL_i)P(D_{i-3}) \\ & + P(P_{i-1})P(P_{i-2})P(K_{i-3})P(D_{i-3}) \\ & + P(P_{i-1})P(P_{i-2})P(P_{i-3})P(K_{i-4})P(D_{i-4}) \\ & + \sum_{j=5}^i [P(P_{i-1}) \dots P(P_{i-j+1})P(K_{i-j})P(D_{i-j})] \\ & + P(P_{i-1})P(P_{i-2}) \dots P(P_0). \end{aligned}$$

Note that the 1st, 2nd,  $\dots$ , 7th terms in the above sum correspond to the right-hand side (RHS) of Eqs. (1)–(7), respectively. The summation term has the index  $j$  ranging from 5 to  $i$ , because the RHS of Eq. (8) is repeated for  $j = 5, \dots, i$ . The last term corresponds to the RHS of Eq. (9).

By merging the product terms of similar structures in the above equation and setting  $P(D_j)$  to  $d_j$ , we can further simplify the above equation as

$$\begin{aligned} d_i = & \sum_{j=1}^2 P(P_{i-1}) \dots P(P_{i-j+1})P(G_{i-j})d_{i-j} \\ & + \sum_{j=1}^i P(P_{i-1}) \dots P(P_{i-j+1})P(K_{i-j})d_{i-j} \\ & + P(P_{i-1})P(P_{i-2})P(GL_i)d_{i-3} + P(P_{i-1}) \dots P(P_0). \end{aligned}$$

Note that the above equation is for a case where  $t_i = 2$ . For an arbitrary  $t_i$ , the expression for  $d_i$  can be obtained by simply replacing the value 2 in the above equation with  $t_i$ . It is

$$\begin{aligned} d_i = & \sum_{j=1}^{t_i} P(P_{i-1}) \dots P(P_{i-j+1})P(G_{i-j})d_{i-j} \\ & + \sum_{j=1}^i P(P_{i-1}) \dots P(P_{i-j+1})P(K_{i-j})d_{i-j} \quad (10) \\ & + P(P_{i-1}) \dots P(P_{i-t_i})P(GL_i)d_{i-t_i-1} \\ & + P(P_{i-1}) \dots P(P_0). \end{aligned}$$

The above equation gives a recursive way to calculate  $d_i$ . Note that to calculate all  $d_i$ 's ( $1 \leq i \leq m - 1$ ), it involves calculating the products  $P(P_i) \dots P(P_j)$  for all  $0 \leq j < i \leq m - 2$ . These products can be computed first and looked up later. Given this optimization, the time complexity for obtaining all  $d_i$ 's is  $O(m^2)$ . Consequently, the time complexity for calculating the ER is also  $O(m^2)$ .

For the special case where  $t_i = 0$ , the first summation term in Eq. (10) disappears and the third term  $P(P_{i-1}) \dots P(P_{i-t_i})P(GL_i)d_{i-t_i-1}$  reduces to  $P(GL_i)d_{i-1}$ . Thus, Eq. (10) becomes

$$\begin{aligned} d_i = & \sum_{j=1}^i P(P_{i-1}) \dots P(P_{i-j+1})P(K_{i-j})d_{i-j} \\ & + P(GL_i)d_{i-1} + P(P_{i-1}) \dots P(P_0). \end{aligned}$$

For the special case where  $k'_i = 0$ , the carry generator of block  $i$  only covers blocks  $(i-1), \dots, (i-t_i)$ . Therefore, the situation shown in Fig. 3(e) does not exist. The situation shown in Fig. 3(f) becomes the situation where  $P_{i-1} = P_{i-2} = G_{i-3} = 1$ . With a simple analysis, we find that the only change for calculating  $d_i$  is to set  $P(GL_i)$  to 0, which turns Eq. (10) into

$$\begin{aligned} d_i &= \sum_{j=1}^{t_i} P(P_{i-1}) \cdots P(P_{i-j+1}) P(G_{i-j}) d_{i-j} \\ &\quad + \sum_{j=1}^i P(P_{i-1}) \cdots P(P_{i-j+1}) P(K_{i-j}) d_{i-j} \\ &\quad + P(P_{i-1}) \cdots P(P_0). \end{aligned}$$

## 5 THEORY FOR OBTAINING THE ERROR DISTRIBUTION

In this section, we show the theory for obtaining the error distribution. For a given input combination, the output error is defined as  $err = (c_o^* s_{n-1}^* \cdots s_0^*)_2 - (c_o s_{n-1} \cdots s_0)_2$ , where  $c_o, s_{n-1}, \dots, s_0$  are the correct output bits of the adder. By definition, the error distance is the absolute value of the error (i.e.,  $|err|$ ). The error distribution is the probability distribution of the error distances.

The binary representation of an error distance is called an error pattern. For example, 0001 0001 0000 is a possible error pattern for a 12-bit block-based approximate adder with 3 blocks of equal length of 4. It is easily seen that there is a one-to-one correspondence between error patterns and error distances. Therefore, the basic idea of our method for obtaining the error distribution is to find out all possible error patterns and then calculate their occurring probabilities.

We observe that not every bit position in an error pattern could have a 1 (corresponding to an error at that bit position). In what follows, we will first study what locations in an error pattern could have a 1. The related results will be described in Section 5.1. Suppose the set of positions in an error pattern where a 1 could occur is  $S_p$ . We also find that not every combination of positions from the set  $S_p$  can give a valid error pattern. To determine the position combinations that give valid error patterns, we next study a basic problem: given that there is a 1 at position  $i \in S_p$  in the error pattern, where the next 1 on the left of  $i$  could occur in the error pattern and what its probability is. The related results will be described in Section 5.2. Finally, with the answer to that basic problem, we present how to obtain the position combinations that give valid error patterns and their probabilities in Section 5.3.

### 5.1 The Positions of 1s in Error Pattern

In this section, we study the following problem: at which positions in an error pattern could a 1 possibly occur? The following claim gives an answer to this question.

#### Theorem 1

A 1 in an error pattern can only occur at position  $sl_i$ , where  $1 \leq i \leq m-1$ . A necessary and sufficient condition for a 1 to occur at position  $sl_i$  is  $c_i = 1$ ,  $c_i^* = 0$ , and the condition that  $P_{i-1} = 1$  and  $c_{i-1}^* \neq c_{i-1}$  is not satisfied.

The proof of this theorem can be found in Appendix A. Here, we give an intuitive explanation for this theorem. By the design of the block-based approximate adder, a 1 in an

error pattern can only occur at position  $sl_i$ , where  $1 \leq i \leq m-1$ . If it occurs at position  $sl_i$ , a necessary condition is that the correct carry-in to block  $i$ ,  $c_i$ , is not equal to the speculated carry-in,  $c_i^*$ . Note that by the design, the only situation for  $c_i \neq c_i^*$  is  $c_i = 1$  and  $c_i^* = 0$ . However, this condition is only necessary. When  $P_{i-1} = 1$ ,  $c_{i-1}^* \neq c_{i-1}$ , we can derive that  $c_{i-1}^* = c_i^* = 0$  and  $c_{i-1} = c_i = 1$ . Although in this case, we have  $c_i = 1$  and  $c_i^* = 0$ , it is not possible to have a 1 at position  $sl_i$  of the error pattern. Excluding this special situation, when  $c_i = 1$  and  $c_i^* = 0$ , bit position  $sl_i$  in an error pattern is guaranteed to have a 1.

Another useful result that we will use later is the following claim. It gives a necessary condition for a 1 to occur at position  $sl_i$  of the error pattern in terms of the group propagate and generate signals.

#### Theorem 2

If there is a 1 at position  $sl_i$  ( $1 \leq i \leq m-1$ ) of the error pattern, then  $P_{i-1} = \cdots = P_{i-t_i} = PL_i = 1$  and  $G_{(cl_i-1):0} = 1$ .

The proof can be found in Appendix B.

### 5.2 Successive Error Positions

From the previous section, we know that 1s in an error pattern could possibly occur at bit positions  $sl_i$  ( $1 \leq i \leq m-1$ ). However, given an approximate adder, not every combination of positions  $sl_i$  gives a valid error pattern for that adder. Therefore, in order to avoid the unnecessary computation, we will next characterize what position combinations could give valid error patterns. As a first step to answer this question, we study what are the possible successive error positions in an error pattern. More specifically, we consider the event that given a 1 at position  $sl_j$  in an error pattern, the next 1 on the left of  $sl_j$  is at position  $sl_i$ , where  $0 \leq j < i < m$ . We define such an event as  $E_{i,j}$ . Note that when  $j = 0$ , since position  $sl_0$  in an error pattern cannot have a 1,  $E_{i,j}$  reduces to the event that  $sl_i$  is the rightmost position in an error pattern that has a 1. We define  $e_{i,j}$  as the occurring probability of event  $E_{i,j}$ . In the rest of this section, we focus on identifying the conditions for event  $E_{i,j}$  to occur and calculating its probability  $e_{i,j}$ , for each pair of  $i$  and  $j$  with  $0 \leq j < i < m$ .

First, we have the following two claims on event  $E_{i,j}$ .

#### Theorem 3

If  $cl_i = cl_{i-1}$ , where  $cl_i$  and  $cl_{i-1}$  denote the lowest bit position of carry generator  $CG_i$  and  $CG_{i-1}$ , respectively, then event  $E_{i,j}$  cannot occur and hence,  $e_{i,j} = 0$ .

#### Theorem 4

If  $i-j \leq t_i$ , then event  $E_{i,j}$  cannot occur and hence,  $e_{i,j} = 0$ .

Their proofs can be found in Appendices C and D, respectively. Theorem 4 is an important theorem, which helps us avoid checking many invalid error patterns. Specifically, if  $i-j \leq t_i$ , then there is no error pattern that can have 1s at positions  $sl_i$  and  $sl_j$ .

By Theorems 3 and 4, we have the following claim.

#### Corollary 1

If event  $E_{i,j}$  ( $0 \leq j < i < m$ ) occurs, then we must have  $cl_i > cl_{i-1}$  and  $i-j > t_i$ .

In the following analysis, we will only consider the case where  $cl_i > cl_{i-1}$  and  $i-j > t_i$ .

By our analysis, to obtain the conditions for  $E_{i,j}$  to occur, we need to distinguish two cases based on whether  $cl_{i-1} \geq sl_j$ , where  $cl_{i-1}$  is the position of the lowest bit of carry generator  $CG_{i-1}$  and  $sl_j$  is the position of the lowest bit of block  $j$ . We discuss these two cases separately in the following two subsections.

### 5.2.1 The Case where $cl_{i-1} \geq sl_j$

In this section, we analyze event  $E_{i,j}$  for the case where  $cl_{i-1} \geq sl_j$ . Note that we are analyzing under the assumption that  $cl_i > cl_{i-1}$ . Thus,  $cl_i - 1 \geq cl_{i-1}$ . The key of our analysis is to focus on the input bits from position  $(cl_i - 1)$  to  $cl_{i-1}$ . First, we have the following claim.

#### Lemma 1

When  $cl_i > cl_{i-1}$ ,  $i - j > t_i$ , and  $cl_{i-1} \geq sl_j$ , if event  $E_{i,j}$  occurs, then we have  $G_{(cl_i-1):cl_{i-1}} = 1$ .

The proof can be found in Appendix E. Lemma 1 means that if event  $E_{i,j}$  occurs, the input bits from position  $(cl_i - 1)$  to  $cl_{i-1}$  should let the group generate signal  $G_{(cl_i-1):cl_{i-1}}$  be 1. Note that the bits from position  $(cl_i - 1)$  to  $cl_{i-1}$  could be in the same block or across multiple blocks. Our further analysis will distinguish these two cases.

The first case is that the bits from position  $(cl_i - 1)$  to  $cl_{i-1}$  cross multiple blocks. We use the example shown in Fig. 4 to illustrate how we analyze this case. In this example,  $t_i = 1$  and  $t_{i-1} = 2$ . Therefore,  $CG_i$  covers block  $(i - 1)$  and the left  $k'_i$  bits of block  $(i - 2)$ , while  $CG_{i-1}$  covers block  $(i - 2)$ , block  $(i - 3)$ , and the left  $k'_{i-1}$  bits of block  $(i - 4)$ . It can be seen that the bits from position  $(cl_i - 1)$  to  $cl_{i-1}$  cross multiple blocks.

Since event  $E_{i,j}$  occurs, there is a 1 at position  $sl_i$  of the error pattern. By Theorem 2, we have  $P_{i-1} = PL_i = 1$ . By Lemma 1, we also have  $G_{(cl_i-1):cl_{i-1}} = 1$ . As shown in Fig. 4(a), the bits from position  $(cl_i - 1)$  to position  $cl_{i-1}$  cover the right part of block  $(i - 2)$ , the entire block  $(i - 3)$ , and the left part of block  $(i - 4)$ . Thus, the condition  $G_{(cl_i-1):cl_{i-1}} = 1$  can be split into the following three cases:

- 1) The case where  $GR_i = 1$ , as shown in Fig. 4(a).
- 2) The case where  $PR_i = G_{i-3} = 1$ , as shown in Fig. 4(b).
- 3) The case where  $PR_i = P_{i-3} = GL_{i-1} = 1$ , as shown in Fig. 4(c).

For case 1, since  $P_{i-1} = PL_i = GR_i = 1$ , we have  $c_i = 1$  and  $c_i^* = 0$ . Furthermore, since  $PL_i = GR_i = 1$  and carry generator  $CG_{i-1}$  covers the entire bits in block  $(i - 2)$ , we have  $c_{i-1} = c_{i-1}^* = 1$ . By Theorem 1, there is a 1 at position  $sl_i$  of the error pattern. Also, given that  $c_{i-1} = c_{i-1}^* = 1$ , by Theorem 1, there is no 1 at position  $sl_{i-1}$  of the error pattern. Thus, event  $E_{i,j}$  occurs if and only if the following event occurs: given that there is a 1 at position  $sl_j$  of the error pattern, there are no 1s at positions  $sl_{i-2}, \dots, sl_{j+1}$  of the error pattern. For simplicity, we denote this event as  $H_{i-2,j}$ . Next, we analyze when event  $H_{i-2,j}$  occurs.

We consider a pair of inputs that causes event  $H_{i-2,j}$  to occur. Since position  $sl_j$  of the error pattern has a 1, by Theorem 1, the correct carry-in to block  $j$  is  $c_j = 1$ . Thus, the correct sum from block  $(i - 2)$  to  $j$  is

$$(s_{sl_{i-1}-1} \dots s_{sl_j})_2 = (a_{sl_{i-1}-1} \dots a_{sl_j})_2 + (b_{sl_{i-1}-1} \dots b_{sl_j})_2 + 1, \quad (11)$$

where  $a_{sl_{i-1}-1}, \dots, a_{sl_j}$  and  $b_{sl_{i-1}-1}, \dots, b_{sl_j}$  denote the bits from block  $(i - 2)$  to  $j$  of the two inputs, respectively.

On the other hand, since there are no 1s at positions  $sl_{i-2}, \dots, sl_{j+1}$  of the error pattern and there is a 1 at

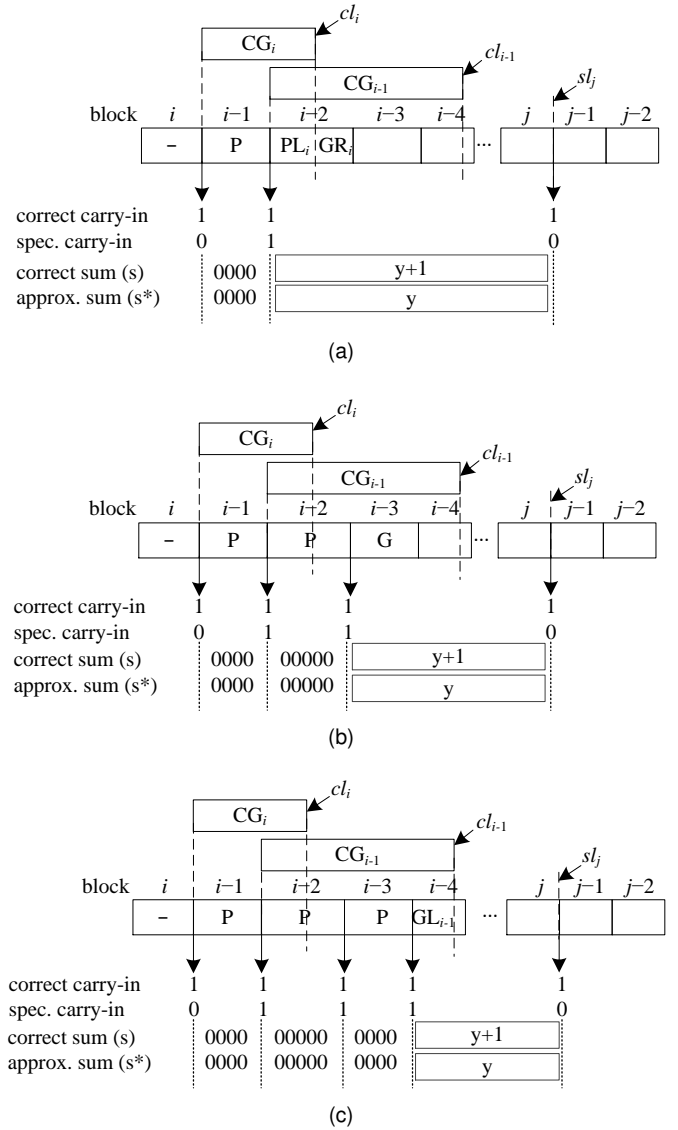


Fig. 4: An example approximate adder with  $cl_{i-1} \geq sl_j$  and the bits from position  $(cl_i - 1)$  to  $cl_{i-1}$  across multiple blocks.

positions  $sl_j$  of the error pattern, the correct sum from block  $(i - 2)$  to  $j$  is larger than the approximate sum by 1, i.e.,

$$(s_{sl_{i-1}-1} \dots s_{sl_j})_2 = (s_{sl_{i-1}-1}^* \dots s_{sl_j}^*)_2 + 1,$$

where  $(s_{sl_{i-1}-1}^* \dots s_{sl_j}^*)_2$  denotes the approximate sum from block  $(i - 2)$  to  $j$ .

Given Eq. (11), the approximate sum is

$$(s_{sl_{i-1}-1}^* \dots s_{sl_j}^*)_2 = (a_{sl_{i-1}-1} \dots a_{sl_j})_2 + (b_{sl_{i-1}-1} \dots b_{sl_j})_2. \quad (12)$$

To proceed, we first introduce an  $(i, j)$  imaginary approximate adder. It is an  $(i - j + 1)$ -block approximate adder constructed from blocks  $i, \dots, j$  ( $0 \leq j \leq i < m$ ) of the original approximate adder. Block  $r$  ( $0 \leq r \leq i - j$ ) of the imaginary approximate adder has almost the same sub-adder and carry generator as block  $(r + j)$  of the original approximate adder. The only difference is that for any  $0 \leq r \leq i - j$ , if the carry generator of block  $(r + j)$  of the original adder covers some bits on the right of the lowest bit of block  $j$ , we will truncate the carry generator

of block  $r$  of the imaginary adder at bit position 0. Note that this truncation is necessary, since otherwise, the carry generator of block  $r$  of the imaginary adder will go beyond bit position 0. This imaginary approximate adder has the following property.

**Theorem 5**

Suppose a pair of inputs causes a 1 at position  $sl_j$  of the error pattern of the original approximate adder. Assume the bits from block  $i$  to  $j$  ( $i \geq j$ ) in the two inputs are  $a_{sl_{i+1}-1}, \dots, a_{sl_j}$  and  $b_{sl_{i+1}-1}, \dots, b_{sl_j}$ , respectively. Assume the corresponding approximate sum from block  $i$  to  $j$  is  $(s_{sl_{i+1}-1}^* \dots s_{sl_j}^*)_2$ . Suppose we feed the inputs  $a_{sl_{i+1}-1}, \dots, a_{sl_j}$  and  $b_{sl_{i+1}-1}, \dots, b_{sl_j}$  to the  $(i, j)$  imaginary approximate adder constructed from the original approximate adder. Then, the sum of the imaginary approximate adder is equal to  $(s_{sl_{i+1}-1}^* \dots s_{sl_j}^*)_2$ .

The proof can be found in Appendix F. Combining the above theorem with Eq. (12), we conclude that a pair of inputs causes event  $H_{i-2,j}$  to occur if and only if the input bits from block  $(i-2)$  to  $j$  cause the  $(i-2, j)$  imaginary approximate adder to produce the correct sum. This further indicates that the input bits from block  $(i-3)$  to  $j$  should make all the speculated carry-ins  $c_{i-j-2}^*, \dots, c_0^*$  of the  $(i-2, j)$  imaginary approximate adder be correct. We denote this event as  $D_{i-2,j}$  and its occurring probability as  $d_{i-2,j}$ . The value of  $d_{i-2,j}$  can be obtained by applying the method described in Section 4 on the imaginary adder.

In summary, for event  $E_{i,j}$  to occur under case 1, we require that  $P_{i-1} = PL_i = GR_i = 1$  and that the inputs from block  $(i-3)$  to  $j$  should make event  $D_{i-2,j}$  occur. Thus, the probability for event  $E_{i,j}$  to occur under case 1 can be calculated as

$$P_a = P(P_{i-1})P(PL_i)P(GR_i)d_{i-2,j}. \quad (13)$$

Case 2 is shown in Fig. 4(b). It has  $P_{i-1} = P_{i-2} = G_{i-3} = 1$ . By a similar argument as case 1, we can show that  $c_i = 1$ ,  $c_i^* = 0$ ,  $c_{i-1} = c_{i-1}^* = 1$ , and  $c_{i-2} = c_{i-2}^* = 1$ . By Theorem 1, there is a 1 at position  $sl_i$  of the error pattern and there are no 1s at positions  $sl_{i-1}$  and  $sl_{i-2}$  of the error pattern. Thus, event  $E_{i,j}$  occurs if and only if event  $H_{i-3,j}$  occurs. By the same argument used in the analysis of case 1, event  $H_{i-3,j}$  occurs if and only if the inputs from block  $(i-4)$  to  $j$  make event  $D_{i-3,j}$  occur. Thus, the probability for event  $E_{i,j}$  to occur under case 2 can be calculated as

$$P_b = P(P_{i-1})P(P_{i-2})P(G_{i-3})d_{i-3,j}. \quad (14)$$

Case 3 is shown in Fig. 4(c). By a similar argument as cases 1 and 2, we can show that the probability for event  $E_{i,j}$  to occur under case 3 can be calculated as

$$e_{i,j} = P(P_{i-1})P(P_{i-2})P(P_{i-3})P(GL_{i-1})d_{i-4,j}. \quad (15)$$

Combining Eqs. (13), (14), and (15), we obtain that the value of  $e_{i,j}$  for the example adder shown in Fig. 4, where  $cl_{i-1} \geq sl_j$ ,  $t_i = 1$ , and  $t_{i-1} = 2$ , is

$$\begin{aligned} e_{i,j} &= P(P_{i-1})P(PL_i)P(GR_i)d_{i-2,j} \\ &+ P(P_{i-1})P(P_{i-2})P(G_{i-3})d_{i-3,j} \\ &+ P(P_{i-1})P(P_{i-2})P(P_{i-3})P(GL_{i-1})d_{i-4,j}. \end{aligned}$$

For any arbitrary  $t_i$  and  $t_{i-1}$ , as long as the bits from position  $(cl_i - 1)$  to  $cl_{i-1}$  cross multiple blocks, we can generalize the above analysis and obtain that

$$\begin{aligned} e_{i,j} &= P(P_{i-1}) \cdots P(P_{i-t_i})P(PL_i)P(GR_i)d_{i-t_i-1,j} \\ &+ \sum_{q=1}^{t_{i-1}-t_i} P(P_{i-1}) \cdots P(P_{i-t_i-q})P(G_{i-t_i-q-1})d_{i-t_i-q-1,j} \\ &+ P(P_{i-1}) \cdots P(P_{i-t_{i-1}-1})P(GL_{i-1})d_{i-t_{i-1}-2,j}. \end{aligned} \quad (16)$$

Note that in Eq. (16), the second term  $\sum_{q=1}^{t_{i-1}-t_i} P(P_{i-1}) \cdots P(P_{i-t_i-q})P(G_{i-t_i-q-1})d_{i-t_i-q-1,j}$  is a sum of  $(t_{i-1} - t_i)$  products. Its first term is  $P(P_{i-1}) \cdots P(P_{i-t_{i-1}})P(G_{i-t_{i-1}-2})d_{i-t_{i-1}-3,j}$  and its last term is  $P(P_{i-1}) \cdots P(P_{i-t_{i-1}-1})P(GL_{i-1})d_{i-t_{i-1}-2,j}$ . In the example of Fig. 4,  $t_i = 1$ ,  $t_{i-1} = 2$ , and  $t_{i-1} - t_i = 1$ . Therefore, this sum only contains one product, i.e.,  $P(P_{i-1})P(P_{i-2})P(G_{i-3})d_{i-3,j}$ . Note that when the bits from position  $(cl_i - 1)$  to  $cl_{i-1}$  cross multiple blocks, it can be proved that  $t_{i-1} - t_i \geq 0$ . In the special case where  $t_{i-1} - t_i = 0$ , the summation term does not exist.

The example in Fig. 4 has  $k'_i > 0$  and  $k'_{i-1} > 0$ . In the special case where  $k'_i = 0$ , the segment labeled  $PL_i$  in Fig. 4(a) does not exist and the segment labeled  $GR_i$  becomes the entire block  $(i-2)$ . Thus, in the first term of Eq. (16),  $P(PL_i)$  degenerates to 1 and  $P(GR_i)$  becomes  $P(G_{i-t_i-1})$ . Therefore, the first term in Eq. (16) becomes

$$P(P_{i-1}) \cdots P(P_{i-t_i})P(G_{i-t_i-1})d_{i-t_i-1,j}.$$

In the special case where  $k'_{i-1} = 0$ , the segment labeled  $GL_{i-1}$  in Fig. 4(c) does not exist. It degenerates to 0. Therefore, the third term in Eq. (16) does not exist.

The example in Fig. 4 also has  $t_i > 0$ . In the special case where  $t_i = 0$ , the product  $P(P_{i-1}) \cdots P(P_{i-t_i})$  in the first term of Eq. (16) degenerates to 1. Thus, the first term in Eq. (16) becomes  $P(PL_i)P(GR_i)d_{i-1,j}$ .

Now, we consider the second case where the bits from position  $(cl_i - 1)$  to  $cl_{i-1}$  are in the same block. We also illustrate how we analyze this case using an example, which is shown in Fig. 5.

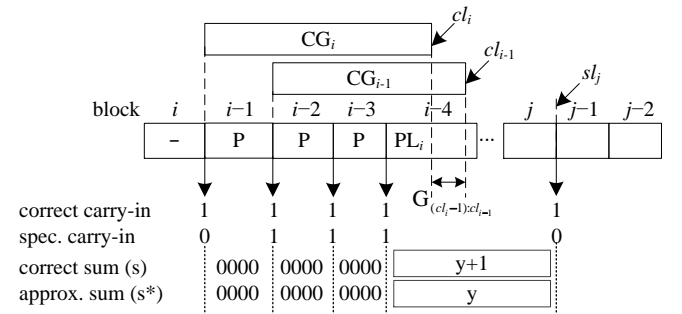


Fig. 5: An example approximate adder with  $cl_{i-1} \geq sl_j$  and the bits from position  $(cl_i - 1)$  to  $cl_{i-1}$  in the same block.

In this example,  $t_i = 3$  and  $t_{i-1} = 2$ . The bits from position  $(cl_i - 1)$  to  $cl_{i-1}$  are all in block  $(i-4)$ . Since event  $E_{i,j}$  occurs, there is a 1 at position  $sl_i$  of the error pattern. By Theorem 2, we have  $P_{i-1} = P_{i-2} = P_{i-3} = PL_i = 1$ . Furthermore, by Lemma 1, we have  $G_{(cl_i-1):cl_{i-1}} = 1$ . By the same analysis as before, we can show that event  $E_{i,j}$  occurs if and only if event  $H_{i-4,j}$  occurs. The occurring probability



of  $H_{i-4,j}$  is  $d_{i-4,j}$ . Thus, the occurring probability of event  $E_{i,j}$  is

$$e_{i,j} = P(P_{i-1})P(P_{i-2})P(P_{i-3})P(PL_i) \\ P(G_{(cl_i-1):cl_{i-1}})d_{i-4,j}.$$

For any arbitrary  $t_i$  and  $t_{i-1}$ , as long as the bits from position  $(cl_i - 1)$  to  $cl_{i-1}$  are in the same block, we can generalize the above analysis and obtain that

$$e_{i,j} = P(P_{i-1})P(P_{i-2}) \cdots P(P_{i-t_i})P(PL_i) \\ P(G_{(cl_i-1):cl_{i-1}})d_{i-t_i-1,j}. \quad (17)$$

### 5.2.2 The Case where $cl_{i-1} < sl_j$

In this section, we analyze event  $E_{i,j}$  for the case where  $cl_{i-1} < sl_j$ . This case is similar to the case where  $cl_{i-1} \geq sl_j$ . The difference lies in that bit position  $sl_j$  now is in between position  $(cl_i - 1)$  and  $cl_{i-1}$ . Given this difference, Lemma 1 changes to the following one.

#### Lemma 2

When  $cl_i > cl_{i-1}$ ,  $i - j > t_i$ , and  $cl_{i-1} < sl_j$ , if event  $E_{i,j}$  occurs, then we have

$$G_{(cl_i-1):sl_j} = 1.$$

The proof is similar to that of Lemma 1. We omit it due to space limit.

Given Lemma 2, the case where  $cl_{i-1} < sl_j$  can be reduced to the case where  $cl_{i-1} = sl_j$ . In other words, when  $cl_{i-1} < sl_j$ , we can pretend that carry generator  $CG_{i-1}$  covers and only covers the entire blocks  $i - 2, \dots, j$ . This special situation belongs to the case where  $cl_{i-1} \geq sl_j$ , which has been analyzed before. Thus, the probability  $e_{i,j}$  can be derived from Eqs. (16) and (17) by assuming that  $cl_{i-1} = sl_j$ . Note that Eq. (16) applies when the bits from position  $(cl_i - 1)$  to  $sl_j$  cross multiple blocks and Eq. (17) applies when these bits are in the same block. Next, we discuss how Eqs. (16) and (17) can be simplified under the assumption that  $cl_{i-1} = sl_j$ .

First, we consider the situation where the bits from position  $(cl_i - 1)$  to  $sl_j$  cross multiple blocks. We apply Eq. (16) to calculate  $e_{i,j}$ . Since we assume that carry generator  $CG_{i-1}$  covers the entire blocks  $i - 2, \dots, j$ , we have that  $k'_{i-1}$  of  $CG_{i-1}$  is 0. Thus, the third term in Eq. (16) does not exist. Furthermore, for this special carry generator, we have  $i - 1 - t_{i-1} = j$ . Thus,  $t_{i-1}$  in Eq. (16) is equal to  $i - 1 - j$ . In summary, the probability  $e_{i,j}$  can be calculated from Eq. (16) by dropping its third term and setting  $t_{i-1}$  to  $i - 1 - j$ . Thus, we have

$$e_{i,j} = P(P_{i-1}) \cdots P(P_{i-t_i})P(PL_i)P(GR_i)d_{i-t_i-1,j} \\ + \sum_{q=1}^{i-1-j-t_i} P(P_{i-1}) \cdots P(P_{i-t_i-q})P(G_{i-t_i-q-1})d_{i-t_i-q-1,j}. \quad (18)$$

Now, we consider the situation where the bits from position  $(cl_i - 1)$  to  $sl_j$  are in the same block. We apply Eq. (17) to calculate  $e_{i,j}$ . Note that bit position  $(cl_i - 1)$  is in block  $(i - t_i - 1)$  and bit position  $sl_j$  is in block  $j$ . Since bit positions  $(cl_i - 1)$  and  $sl_j$  are in the same block, we have  $i - t_i - 1 = j$ . Given this, we also have  $d_{i-t_i-1,j} = d_{j,j} = 1$ . Furthermore, since  $cl_{i-1} = sl_j$ , the term  $P(G_{(cl_i-1):cl_{i-1}})$  in Eq. (17) is just  $P(G_{(cl_i-1):sl_j})$ . Given that positions  $(cl_i - 1)$  and  $sl_j$  are in the same block, by our definitions,  $P(G_{(cl_i-1):sl_j})$  is just  $P(GR_i)$ . Thus, the probability  $e_{i,j}$  can be calculated by

setting  $t_i$  to  $(i - 1 - j)$ ,  $P(G_{(cl_i-1):cl_{i-1}})$  to  $P(GR_i)$ , and  $d_{i-t_i-1,j}$  to 1 in Eq. (17). Thus, we have

$$e_{i,j} = P(P_{i-1})P(P_{i-2}) \cdots P(P_{j+1})P(PL_i)P(GR_i). \quad (19)$$

### 5.3 Valid Error Patterns and their Probabilities

In this section, we finally show that given an approximate adder, what the valid error patterns are and what their probabilities are.

Assume a valid error pattern has 1s at bit positions  $sl_{i_1}, \dots, sl_{i_r}$ , where  $0 < i_1 < i_2 < \cdots < i_r \leq m - 1$ . Assume  $i_0 = 0$ . Then, events  $E_{i_1,i_0}, E_{i_2,i_1}, \dots, E_{i_r,i_{r-1}}$  must occur. By Corollary 1, we must have that for all  $1 \leq j \leq r$ ,  $i_j - i_{j-1} > t_{i_j}$  and  $cl_{i_j} > cl_{i_{j-1}}$ .

Furthermore, the error pattern also implies that the following event must occur: given that there is a 1 at position  $sl_{i_r}$  ( $i_r \leq m - 1$ ), there is no 1 on the left of position  $sl_{i_r}$ . This is just event  $H_{m-1,i_r}$  defined in Section 5.2.

In summary, the error pattern occurs when events  $E_{i_1,i_0}, E_{i_2,i_1}, \dots, E_{i_r,i_{r-1}}$  and  $H_{m-1,i_r}$  occur. The occurring probability of event  $E_{i_j,i_{j-1}}$  is  $e_{i_j,i_{j-1}}$  and that of event  $H_{m-1,i_r}$  is  $d_{m-1,i_r}$ . Since we assume that the inputs are bit-wise independent, the occurring probability of the error pattern is  $d_{m-1,i_r} \cdot \prod_{j=1}^r e_{i_j,i_{j-1}}$ . In summary, we have the following claim on the valid error patterns and their occurring probabilities.

#### Theorem 6

Assume a valid error pattern has 1s at bit positions  $sl_{i_1}, \dots, sl_{i_r}$ , where  $0 < i_1 < i_2 < \cdots < i_r \leq m - 1$ . Assume  $i_0 = 0$ . Then, we must have that for all  $1 \leq j \leq r$ ,  $i_j - i_{j-1} > t_{i_j}$  and  $cl_{i_j} > cl_{i_{j-1}}$ . The probability of the error pattern is

$$d_{m-1,i_r} \cdot \prod_{j=0}^{r-1} e_{i_{j+1},i_j}.$$

## 6 ALGORITHM FOR OBTAINING THE ERROR DISTRIBUTION

In this section, we present the algorithm for efficiently obtaining the error distribution.

Using Theorem 6, we can enumerate all possible error patterns and calculate their probabilities 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. To realize this, we propose a method that grows a partial error pattern and its probability into the complete error pattern and the associated probability. The procedure uses a recursive helper function  $ED(i, j, ePar, pPar)$  shown in Algorithm 1. The argument  $i$  refers to bit position  $sl_i$ , which is under check and can potentially have a 1.  $j$  refers to bit position  $sl_j$ , which is the nearest bit position on the right of  $sl_i$  in the error pattern that has a 1.  $ePar$  is the partial error distance and  $pPar$  is the partial probability.

When checking bit position  $sl_i$ , we have the following two cases:

- 1) There is a 1 at position  $sl_i$  in the error pattern. Then, the nearest bit position on the left of  $sl_i$  that could have a 1 is  $sl_h$ , where  $h$  is the smallest integer satisfying that  $i + t_h < h < m$  and  $cl_h > cl_{h-1}$ . We continue to check that bit position. The error magnitude  $2^{sl_i}$  is added to

the partial error distance and the partial probability is multiplied with the value  $e_{i,j}$ . This is shown in Line 10 of Algorithm 1. Note that in the special case where the smallest integer  $h$  satisfying that  $i + t_h < h < m$  and  $cl_h > cl_{h-1}$  does not exist, we set  $h$  to  $m$  to trigger the termination condition.

- 2) There is no 1 at position  $sl_i$  in the error pattern. Then, we further check bit position  $sl_{i+1}$ , which is the next bit position that could have a 1. The partial error distance and probability do not change. This is shown in Line 11 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:    $pPar \leftarrow pPar \cdot d_{m-1,j}$ ;
3:   print out  $ePar$  and  $pPar$ ;
4:   return ;
5: end if
6: find the smallest  $h$  satisfying that  $i + t_h < h < m$  and
    $cl_h > cl_{h-1}$ ;
7: if  $h$  does not exist then
8:    $h \leftarrow m$ ;
9: end if
10:  $ED(h, i, ePar + 2^{sl_i}, pPar \cdot e_{i,j})$ ;
11:  $ED(i + 1, j, ePar, pPar)$ ;
12: return ;

```

---

When  $i \geq m$ , it is the termination case (see Lines 1-5). In this case,  $ePar$  becomes the complete error distance. The probability of  $ePar$  should be  $pPar$  multiplied by  $d_{m-1,j}$ , which accounts for the probability that there is no 1 on the left of position  $sl_j$  in the error pattern. Then, we print out the error distance  $ePar$  and its probability  $pPar$ . The initial function call is  $ED(i_0, j_0, ePar_0, pPar_0)$ , where  $j_0 = 0$ ,  $ePar_0 = 0$ ,  $pPar_0 = 1$ , and  $i_0$  is the smallest integer satisfying that  $t_{i_0} < i_0 < m$  and  $cl_{i_0} > cl_{i_0-1}$ .

Now, we analyze the time complexity of the algorithm. As shown in Algorithm 1, the algorithm is a recursive procedure. During the recursion, the computation of  $d_{m-1,j}$  ( $0 \leq j \leq m-1$ ),  $e_{i,j}$  ( $0 \leq j \leq m-1, j+t_i < i \leq m-1$ ), and  $h$  will be repeated multiple times. Therefore, we will calculate and store them in a table in advance and look up them from the table during the recursion. We first analyze the time complexity of calculating them in advance. It can be seen that the calculation of the  $h$  values for all  $0 \leq i \leq m-1$  takes  $O(m^2)$  time. The calculation of  $e_{i,j}$ 's involves the calculation of the products  $P(P_i) \cdots P(P_j)$  for all  $0 \leq j < i \leq m-2$  and  $d_{i,j}$ 's for all  $0 \leq j < i \leq m-2$ . We assume that the products  $P(P_i) \cdots P(P_j)$  are computed first and looked up later. From the time complexity analysis in Section 4, the calculation of all  $d_{i,j}$ 's ( $0 \leq j < i \leq m-2$ ) takes  $O(m^3)$  time. With the products  $P(P_i) \cdots P(P_j)$  and  $d_{i,j}$ 's calculated in advance, by Eqs. (16)–(19), the calculation of each  $e_{i,j}$  takes  $O(m)$  time. Since there are at most  $m^2$   $e_{i,j}$ 's that needs to be calculated in advance, the total runtime of calculating all  $e_{i,j}$ 's is  $O(m^3)$ . Furthermore, the calculation of  $d_{m-1,j}$ 's for all  $0 \leq j \leq m-1$  takes  $O(m^2)$  time. In summary, the computation of  $d_{m-1,j}$ 's for all  $0 \leq j \leq m-1$ ,  $e_{i,j}$ 's for all  $0 \leq j \leq m-1$  and  $j+t_i < i \leq m-1$ , and all  $h$  values takes  $O(m^3)$  time.

Now, we analyze the runtime of the recursive procedure shown in Algorithm 1, assuming that all  $d_{m-1,j}$ 's,  $e_{i,j}$ 's, and  $h$  values are known in advance. We can see that each call of the function  $ED$  takes constant time. Thus, the runtime of the recursive procedure is proportional to the number of calls of the function  $ED$ . Now, we analyze the number of

function calls. 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. Each leaf of the tree corresponds to a valid error pattern. Assume that the number of error patterns for the given approximate adder is  $N$ . Then, the recursion tree has  $N$  leaves and  $(N-1)$  internal nodes. Thus, the total number of nodes in the tree is  $(2N-1)$ . Since each call of  $ED$  corresponds to a node in the binary tree, the total number of function calls is  $(2N-1)$ .

In summary, combined with the runtime for calculating  $d_{m-1,j}$ 's,  $e_{i,j}$ 's, and  $h$ 's, the total runtime of the proposed method for obtaining the error distribution is  $O(N + m^3)$ . Since the number of error patterns is usually an exponential function on  $m$ , thus, the time complexity of the proposed method for obtaining the error distribution is  $O(N)$ . On the other hand, any algorithm for obtaining the error distribution must have runtime complexity that is 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.

## 7 CONDITION FOR CORRECTNESS

In the derivation of the methods for calculating the ER and the error distribution, we assume that each input is bit-wise independent. However, this assumption is stricter than what is needed. In fact, the correctness of the methods can still be guaranteed under a more relaxed condition. In this section, we describe such a condition.

Specifically, for calculating ER, we claim that our method is accurate as long as both inputs are block-wise independent. This condition means the random input  $A$  satisfies that the inputs in each block of  $A$  are independent of inputs in any other blocks of  $A$  and the same for the random input  $B$ . The claim can be proven by checking Eq. (10) for calculating  $d_i$ . If both inputs are block-wise independent, then the multiplication of the marginal probabilities in Eq. (10) is equal to the accurate joint probability. For example,  $P(P_{i-1}) \cdots P(P_{i-j+1})P(G_{i-j})d_{i-j} = P(P_{i-1} = 1, \dots, P_{i-j+1} = 1, G_{i-j} = 1, D_{i-j})$ . Note that under this condition, for the special product of probabilities  $P(P_{i-1}) \cdots P(P_{i-t_i})P(GL_i)d_{i-t_i-1}$  in Eq. (10), it is also equal to the accurate joint probability  $P(P_{i-1} = 1, \dots, P_{i-t_i} = 1, GL_i = 1, D_{i-t_i-1})$ . Thus,  $d_i$  calculated by Eq. (10) is accurate and hence, the ER.

For obtaining error distribution, when  $k'_i = 0$  for all  $0 \leq i \leq m-1$ , the condition that guarantees the correctness is same as that for the error rate, i.e., the block-wise independence of both inputs. However, when  $k'_i > 0$  for some  $0 \leq i \leq m-1$ , extra conditions are required besides block-wise independence. This can be seen from Eqs. (16)–(19). From the first term  $P(P_{i-1}) \cdots P(P_{i-t_i})P(PL_i)P(GR_i)d_{i-t_i-1,j}$  in Eq. (16), we additionally require that the left  $k'_i$  inputs of block  $(i-t_i-1)$  of  $A$  and the right  $(k_{i-t_i-1} - k'_i)$  inputs of block  $(i-t_i-1)$  of  $A$  are independent and the same for  $B$ . Once this additional requirement is satisfied, Eqs. (18) and (19) are also accurate. From Eq. (17), we additionally require that for those  $i$  such that  $k'_i > 0$  and positions  $(cl_i-1)$  and  $cl_{i-1}$  are in the same block, the left  $k'_i$  inputs of block  $(i-t_i-1)$  of  $A$  and the inputs from position  $(cl_i-1)$  to  $cl_{i-1}$  of  $A$  are independent and the same for  $B$ .

## 8 EXPERIMENTAL RESULTS

We implemented our methods for calculating error rate and error distribution using C++ and applied them to a set

of block-based approximate adders to study their performances. All the experiments were conducted on a 3.6 GHz desktop running Linux operating system. Next, we will study the accuracy of the proposed method in Section 8.1 and compare the runtime of the proposed method to other existing methods in Section 8.2.

## 8.1 Accuracy Study

In this section, we studied the accuracy of our proposed methods. We conducted two experiments depending on whether the inputs are uniformly distributed or not.

### 8.1.1 Uniformly Distributed Inputs

In the case that the two inputs are independent and uniformly distributed, each input is bit-wise independent and the error distribution produced by our method is exact. Based on the exact error distribution, we can further obtain the exact error statistics, such as ER, MED, and MSE. In this section, we demonstrate the accuracy of our method in handling uniformly-distributed inputs by comparing it with the method in [27] and the Monte Carlo method, which randomly chooses a subset of input combinations for simulation.

TABLE 1: Relative errors in percentage of the method in [27] and the Monte Carlo method for inputs of uniform distribution.

		[27]	Monte Carlo		
			10K	100K	1M
(ETA-IV)	ER	0.812	0.188	0.064	0.055
$n32k2l1$	MSE	8.573	1.705	0.521	0.161
(ETA-II)	ER	4.658	0.436	0.133	0.041
$n32k2l2$	MSE	1.541	2.439	0.669	0.222
$n32k2l5$	ER	0.142	1.843	0.582	0.211
	MSE	0.037	6.629	2.049	0.813
(ETA-IV)	ER	0.381	0.676	0.189	0.064
$n32k4l2$	MSE	1.254	2.236	0.706	0.231
(ETA-II)	ER	2.331	1.687	0.608	0.195
$n32k4l4$	MSE	0.025	4.771	1.355	0.472
$n32k4l10$	ER	4.093	14.991	4.634	1.619
	MSE	0	32.770	11.240	3.922

Specifically, 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 [27] and the Monte Carlo method with sample sizes as 10K, 100K, and 1M, respectively. Note that the method in work [27] has given an exact analytical expression for MED for uniformly distributed inputs. Therefore, we did not consider the relative errors of MED in this experiment.

The experimental results are shown in Table 1. The experiment was conducted on 6 block-based approximate adders with equal block length. Their configurations are listed in the first column of Table 1. The value after  $n$  gives the size of the adder. The value after  $k$  gives the length of each block. The value after  $l$  gives the length of the carry generator in the normal case. If  $l > sl_i$  for a carry generator  $CG_i$  ( $1 \leq i < m$ ), then the length of  $CG_i$  is  $sl_i$ . For example,  $n32k2l5$  refers to a 32-bit approximate adder with  $k_{15} = \dots = k_0 = 2$ ,  $l_{15} = \dots = l_3 = 5$ ,  $l_2 = 4$ , and  $l_1 = 2$ . If an adder corresponds to an previously proposed approximate adder, its name is also given. The results in the third column show that the relative errors on ERs and MSEs computed by the method in [27] can be up to 4.7% and 8.6%, respectively. The relative errors on ERs and MSEs produced by the Monte Carlo method are shown in columns 4–6. For each sample size  $M$  in the Monte Carlo method, we ran the entire  $M$ -sample simulation 100 times and obtained the relative error for each simulation run. The final relative

error listed is the average relative error over the 100 runs. 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, as will be shown in Section 8.2.

It should be noted that the method in [27] 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 compare the error distributions given by the Monte Carlo method to the exact error distributions produced by the proposed method for uniformly distributed inputs. We chose two adders with equal block length. They are 64-bit ETA-II with  $k = 4$  and  $l = 4$  and a 64-bit block-based approximate adder with  $k = 4$  and  $l = 10$ , where  $k$  refers to the block length and  $l$  refers to the length of the carry generators in normal cases. We found that the error distances for these adders tend to cluster around  $2^{ik}$ . To show the distribution more clearly, we plotted the sum of probabilities over all error distances in the range  $[2^{ik}, 2^{(i+1)k})$  versus  $ik$ . The plots for the two block-based approximate adders are shown in Fig. 6. In the figure, we did not plot the bar for the error distance of 0, since its probability is much larger than those of the other error distances, which would make the other bars very short if plotted. In the figures, the white bars show the accurate error distribution produced by our method, while the red lines on each bar indicate the range of probabilities in 20 Monte Carlo runs, each with 100K samples. From the figure, we can see that when  $l$  is small, the Monte Carlo method can produce a result close to the accurate distribution, but when  $l$  increases, the accuracy of the Monte Carlo method degrades. The reason is that when  $l$  increases, the error probability decreases. For example, as shown in Fig. 6, the error probability of ETA-II is on the scale of 0.01, while the error probability of the 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 and thus, the accuracy of the Monte Carlo method is reduced. In contrast, our method is guaranteed to obtain the exact distribution for uniformly distributed inputs.

### 8.1.2 Non-uniformly Distributed Inputs

In the case of non-uniformly distributed inputs, the condition described in Section 7 that guarantees the correctness of our methods may not hold and hence, the ER and the error distribution produced by our methods may not be exact. In this section, we empirically studied the accuracy of our method for non-uniformly distributed inputs.

Since the Monte Carlo method may not produce exact results, we verified the accuracy of our method by comparing the results of our method with the accurate results produced by the enumeration method, which enumerates all the input combinations and derives the error metrics from the enumerated values. We conducted experiments on a set of block-based approximate adders for inputs of Gaussian and geometric distributions. The probabilities such as  $P(P_i)$ , and  $P(G_i)$  for different blocks are calculated by the method proposed in [31]. For comparison purpose, we also re-implemented the method proposed in [31] and listed their results together.

The results are shown in Table 2. The adders used in this experiment include ACA-II, GeAr, and ETA-IIM with various configurations, as shown in the first column of



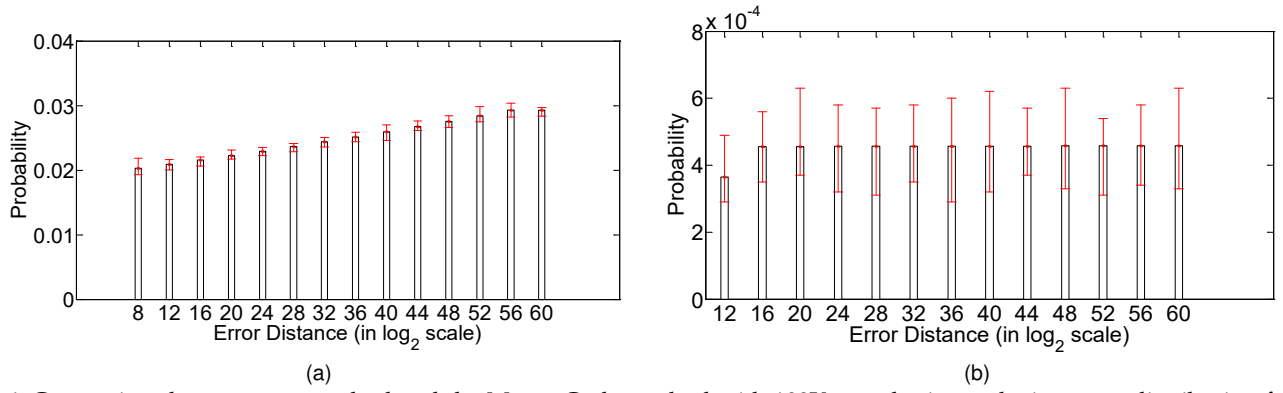


Fig. 6: Comparison between our method and the Monte Carlo method with 100K samples in producing error distribution for: (a) ETA-II with  $k = 4$  and  $l = 4$ ; (b) a block-based approximate adder with  $k = 4$  and  $l = 10$ . Here,  $k$  refers to the block length and  $l$  refers to the length of the carry generators in normal cases.

TABLE 2: Accuracy comparison of the proposed method to the enumeration method and the method in [31] for non-uniformly distributed inputs.

	input distributions		enum	[31] (RE/%)	our (RE/%)
ACA-II	Gaussian( $2^9$ , 100)	ER	0.261	0.273 (4.6)	0.273 (4.6)
		MED	30.15	31.59 (4.8)	31.59 (4.8)
		MSE	6337	6648 (4.9)	6648 (4.9)
$n10k2$	Geometric(0.01)	ER	0.203	0.216 (6.4)	0.216 (6.4)
		MED	16.91	17.86 (5.6)	17.86 (5.6)
		MSE	3042	3211 (5.6)	3211 (5.6)
ACA-II	Gaussian( $2^{11}$ , 900)	ER	0.105	0.100 (-4.8)	0.100 (-4.8)
		MED	30.12	28.86 (-4.2)	28.72 (-4.7)
		MSE	13922	13347 (-4.1)	13273 (-4.7)
$n12k3$	Geometric(0.005)	ER	0.078	0.082 (5.1)	0.082 (5.1)
		MED	17.01	18.19 (7.0)	18.19 (7.0)
		MSE	7230	7783 (7.6)	7783 (7.6)
GeAr	Gaussian( $2^{11}$ , 400)	ER	0.068	0.070 (2.9)	0.070 (2.9)
		MED	30.32	31.61 (4.3)	31.59 (4.2)
		MSE	25246	26325 (4.3)	26304 (4.2)
$n12k2l4$	Geometric(0.01)	ER	0.034	0.035 (2.9)	0.035 (2.9)
		MED	4.558	4.816 (5.7)	4.816 (5.7)
		MSE	896	1010 (12.7)	1010 (12.7)
ETA-IIM	Gaussian( $2^{11}$ , 600)	ER	0.053	0.055 (3.8)	0.055 (3.8)
		MED	3.365	3.491 (3.7)	3.491 (3.7)
		MSE	215.4	223.4 (3.7)	223.4 (3.7)
$n12$	Geometric(0.005)	ER	0.052	0.053 (2.7)	0.053 (2.7)
		MED	3.299	3.418 (3.6)	3.418 (3.6)
		MSE	211.2	218.8 (3.6)	218.8 (3.6)
ETA-IIM	Gaussian( $2^{13}$ , 1000)	ER	0.263	0.272 (3.4)	0.272 (3.4)
		MED	30.38	31.50 (3.7)	31.50 (3.7)
		MSE	6397	6624 (3.5)	6624 (3.5)
$n14$	Geometric(0.001)	ER	0.258	0.269 (4.3)	0.269 (4.3)
		MED	29.46	30.70 (4.2)	30.70 (4.2)
		MSE	6166	6425 (4.2)	6425 (4.2)

Table 2. As we discussed in Section 2, these adders do not have equal block lengths. The values after  $n$  represent the sizes of the adders, which range between 10 and 14. The value after  $k$  for an ACA-II specifies the configuration of the ACA-II (see Section 2 for details about the ACA-II configuration). The values after  $k$  and  $l$  for a GeAr specify the configuration of the GeAr (see Section 2 for details about the GeAr configuration). For the first ETA-IIM in the table, its  $k_0 = 6$ ,  $k_1 = k_2 = 3$ ,  $l_1 = 3$ , and  $l_2 = 6$ . For the second ETA-IIM in the table, its  $k_0 = 4$ ,  $k_1 = \dots = k_5 = 2$ ,  $l_1 = l_2 = l_3 = 2$ ,  $l_4 = 4$ , and  $l_5 = 6$ . The parameters of the input distributions are listed in the second column of Table 2. Gaussian( $\mu, \sigma$ ) represents a Gaussian distribution with mean  $\mu$  and standard deviation  $\sigma$ . Geometric( $p$ ) represents a geometric distribution with probability density function (PDF) as  $P(x = i) = (1 - p)^i p$ . For each test, we assume the two inputs are independent and have the same distribution.

The results of ERs, MEDs, and MSEs produced by the

enumeration method, the method in [31], and our method are shown in the columns titled “enum”, “[31]”, and “our”, respectively. For the method in [31] and our method, we also give the relative errors (REs) in percentage of each method over the enumerate method. The results show that even though the condition of block-wise independence is not strictly satisfied for these non-uniformly distributed inputs, the proposed method could produce the error metrics with high degree of accuracy. Specifically, for ER, the accuracy lies in the range of [93.6%, 97.3%]; for MED, the accuracy lies in the range of [93.0%, 96.4%]; for MSE, except for the test case on GeAr under the geometric distribution, the accuracy lies in the range of [92.4%, 96.5%]. From the results, we also found that the relative errors of the proposed method vary for different input distributions and different approximate adders. We believe the reason is that these factors affect the extent to which we can assume the block-wise independence of the input distribution. Compared to the method in [31], our method produces the identical results for 8 out of the 10 test cases. For the other two test cases, the results of our method and the method in [31] are very close: the differences are no more than 0.6%.

## 8.2 Runtime Study

In this section, we compared the runtime of our methods with the Monte Carlo method and the method proposed in [31].

### 8.2.1 Comparison with the Monte Carlo Method

Fig. 7 shows the runtime of the proposed method and the Monte Carlo methods with 10K, 100K, and 1M samples for obtaining the error distributions. Two approximate adders with equal block length  $k$  and equal carry generator length  $l$  in normal cases were tested. They are ETA-II with  $k = l = 4$  and a block-based approximate adder with  $k = 4$  and  $l = 10$ . We assume that the inputs are uniformly distributed. For each adder, we did experiments on four different block numbers  $m = 4, 8, 12, 16$ . Thus, the adder size  $n$  varies from 16 to 64.

From the two plots in Fig. 7, 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 0.002 seconds. In contrast, for the Monte Carlo method with 10K samples, the runtime is  $0.1 \sim 0.9$  seconds. As the sample size increases, the runtime of the Monte Carlo method also increases linearly. When the sample size



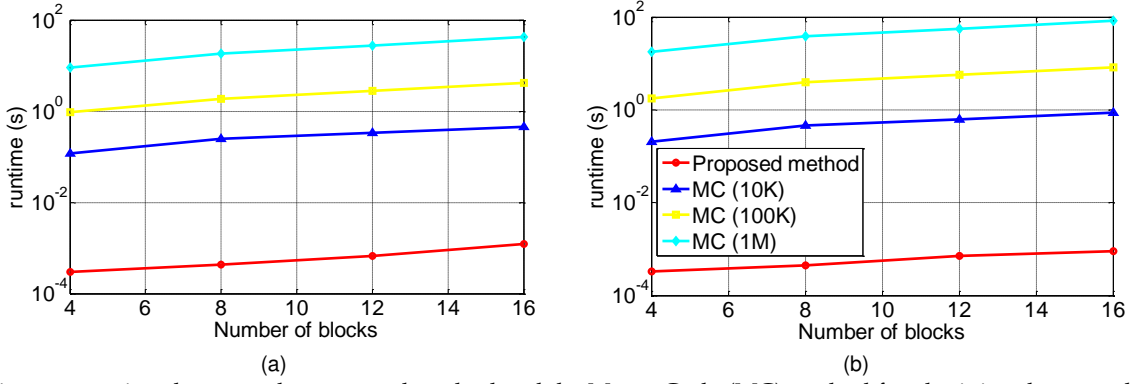


Fig. 7: Runtime comparison between the proposed method and the Monte Carlo (MC) method for obtaining the error distributions for: (a) ETA-II with  $k = 4$  and  $l = 4$ ; (b) a block-based approximate adder with  $k = 4$  and  $l = 10$ . Here,  $k$  refers to the block length and  $l$  refers to the length of the carry generators in normal cases.

reaches 1M, the runtime of the Monte Carlo method is  $10 \sim 90$  seconds.

### 8.2.2 Comparison with the Method in [31]

In this section, we compared the runtime of our methods with the methods proposed in [31]. We compared both the runtime for calculating the ER and that for obtaining the error distribution.

Both our methods and the methods in [31] need to calculate the probabilities such as  $P(P_i)$  and  $P(G_i)$  for different blocks. We first compare the asymptotic time complexity of the methods in [31] and our methods, ignoring the runtime to calculate the probabilities such as  $P(P_i)$  and  $P(G_i)$ . The method for calculating the ER in [31] is based on the inclusion-exclusion principle. It needs to evaluate the probabilities of  $2^{m-1} - 1$  events, where  $m$  is the number of blocks in the adder. Thus, its runtime is  $\Omega(2^m)$ . The method for obtaining the error distribution is even more complex. Our analysis shows that its runtime is  $\Omega(3^m)$ . In contrast, the analysis in Section 4 shows that our method for calculating the ER has time complexity of  $O(m^2)$ . The analysis in Section 6 shows that our method for obtaining the error distribution has time complexity of  $O(N)$ , where  $N$  is number of error patterns. Given an  $m$ -block approximate adder, the only possible locations for a 1 to occur in an error pattern are bit positions  $sl_{m-1}, \dots, sl_1$ . Thus,  $N$  is guaranteed to be no more than  $2^{m-1}$ . Thus, both our method for calculating the ER and that for obtaining the error distribution have a much lower time complexity than the counterparts proposed in [31].

For the entire runtime, both our methods and the methods in [31] also need to include the time for calculating the probabilities such as  $P(P_i)$  and  $P(G_i)$  for different blocks. The runtime for this part further depends on whether the inputs are uniformly distributed or not. In the case of uniform distribution, the computation is fast, since it can be evaluated from a simple expression. Otherwise, the computation takes a much longer time, since it involves the summation of an exponential number of probabilities and the convolution [31]. Next, we empirically study the entire runtime based on whether the inputs are uniformly distributed or not.

Table 3 compares the runtime of our methods and the methods in [31] under uniformly distributed inputs. We used six block-based approximate adders listed in Table 3. The first two are ACA-II's, the next three are GeAr's, and

the last one is an ETA-IIM. The configurations of the ACA-II's and GeAr's are shown in the first column of the table. For the ETA-IIM adder, it has  $k_0 = 8, k_1 = k_2 = \dots = k_{14} = k = 4, l_1 = l_2 = \dots = l_{12} = 4, l_{13} = 8$ , and  $l_{14} = 12$ . We also list the number of blocks  $m$  for each adder in the first column.

From the table, we can see that for calculating the ER, the runtime of the method in [31] roughly increases exponentially with the number of blocks  $m$ . For GeAr ( $n64k3l4$ ) with  $m = 20$ , its runtime is over 200 seconds. In contrast, our method is able to obtain the ER very fast with negligible runtime. For GeAr ( $n64k3l4$ ), our method for calculating the ER achieves a speed-up over  $10^5$  times.

For obtaining the error distribution, the runtime of the method in [31] is much longer than the method in [31] for calculating the ER. It also roughly grows exponentially with the number of blocks  $m$ . However, its growth rate is even faster than that of the method for calculating the ER. For GeAr ( $n64k3l4$ ) with  $m = 20$ , its runtime is over 3.5 hours, indicating that the method in [31] is not practical for approximate adders with large numbers of blocks. In contrast, our method for obtaining the error distribution runs much faster than the method in [31]. For all the adders, our method can obtain the error distribution within 0.3 second. For those adders with at least 15 blocks, our method for obtaining the error distribution achieves a speed-up of at least  $4.8 \times 10^4$  times over the method proposed in [31].

TABLE 3: Runtime for computing the ER and the error distribution using the methods in [31] and the proposed methods for uniformly distributed inputs.

	[31]		proposed	
	error rate(s)	error distr.(s)	error rate(s)	error distr.(s)
ACA-II, $n32k4, m = 7$	0.009	0.109	0	0.003
ACA-II, $n64k4, m = 15$	6.348	2186	0	0.036
GeAr, $n32k3l2, m = 10$	0.13	6.109	0.001	0.021
GeAr, $n64k5l4, m = 12$	0.852	93.586	0.001	0.052
GeAr, $n64k3l4, m = 20$	277	>3.5h	0.002	0.262
ETA-IIM, $n64k4, m = 15$	6.7	2070	0	0.03

Table 4 compares the runtime of our methods and the methods in [31] under Gaussian distributed inputs. We used five block-based approximate adders listed in Table 4. The first two are ACA-II's, the next two are GeAr's, and the last one is an ETA-IIM. The configurations of the ACA-II's and GeAr's are shown in the first column of the table. For the ETA-IIM adder, it has  $k_0 = 6, k_1 = \dots = k_6 = k = 3$ ,

$l_1 = l_2 = l_3 = l_4 = 3$ ,  $l_5 = 6$ , and  $l_6 = 9$ . We also list the number of blocks  $m$  for each adder in the first column.

For the case of non-uniformly distributed inputs, the runtime of our methods increases due to the need to calculate the probabilities such as  $P(P_i)$  and  $P(G_i)$  for different blocks. However, our methods for calculating the ER and the error distribution still achieve up to 900 and 400 times speed-up over the counterparts in [31], respectively.

TABLE 4: Runtime for computing the ER and the error distribution using the methods in [31] and the proposed methods for Gaussian distributed inputs.

	[31]		proposed	
	error rate(s)	error distr.(s)	error rate(s)	error distr.(s)
ACA-II, $n20k4$ , $m = 4$	12.539	12.867	0.602	0.741
ACA-II, $n24k4$ , $m = 5$	3596	3665	13.824	17.493
GeAr, $n16k4l0$ , $m = 4$	10.656	10.764	0.03	0.03
GeAr, $n20k3l2$ , $m = 6$	914	919	1.028	2.417
ETAIM, $n24$ , $m = 7$	62.454	63.346	17.896	19.392

## 9 CONCLUSION

In this paper, we proposed an efficient method for obtaining the error rate and error distribution of block-based approximate adders. The method for obtaining error distribution 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. **Our method gives the exact results for inputs that are block-wise independent, e.g., uniformly distributed inputs.** Experimental results demonstrated that the proposed methods are efficient and accurate. **Compared to the state-of-the-art analytical method, our method is much faster with the same accuracy.**

## 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] M. Shafique, R. Hafiz, S. Rehman *et al.*, "Invited: Cross-layer approximate computing: from logic to architectures," in *Design Automation Conference*, 2016, pp. 99:1–99:6.
- [3] Z. Vasicek and L. Sekanina, "Evolutionary approach to approximate digital circuits design," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 432–444, 2015.
- [4] S. Lee, D. Lee, K. Han *et al.*, "Statistical quality modeling of approximate hardware," in *International Symposium on Quality Electronic Design*, 2016, pp. 163–168.
- [5] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-Simplify: A unified design paradigm for approximate and quality configurable circuits," in *Design, Automation and Test in Europe*, 2013, pp. 796–801.
- [6] D. Palomino, M. Shafique, A. Susin *et al.*, "Thermal optimization using adaptive approximate computing for video coding," in *Design, Automation and Test in Europe*, 2016, pp. 1207–1212.
- [7] F. Sampaio, M. Shafique, B. Zatt *et al.*, "Approximation-aware multi-level cells STT-RAM cache architecture," in *International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 2015, pp. 79–88.
- [8] Y. Wu and W. Qian, "An efficient method for multi-level approximate logic synthesis under error rate constraint," in *Design Automation Conference*, 2016, pp. 1–6.
- [9] H. A. Almurib, T. N. Kumar, F. Lombardi *et al.*, "Inexact designs for approximate low power addition by cell replacement," in *Design, Automation and Test in Europe*, 2016, pp. 660–665.
- [10] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie *et al.*, "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.
- [11] V. Gupta, D. Mohapatra, A. Raghunathan *et al.*, "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.
- [12] A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: A new paradigm for arithmetic circuit design," in *Design, Automation and Test in Europe*, 2008, pp. 1250–1255.
- [13] 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.
- [14] 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.
- [15] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Design Automation Conference*, 2012, pp. 820–825.
- [16] 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.
- [17] R. Ye, T. Wang, F. Yuan *et al.*, "On reconfiguration-oriented approximate adder design and its application," in *International Conference on Computer-Aided Design*, 2013, pp. 48–54.
- [18] M. Shafique, W. Ahmad, R. Hafiz *et al.*, "A low latency generic accuracy configurable adder," in *Design Automation Conference*, 2015, pp. 1–6.
- [19] K. Y. Kyaw, W. L. Goh, and K. S. Yeo, "Low-power high-speed multiplier for error-tolerant application," in *International Conference of Electron Devices and Solid-State Circuits*, 2010, pp. 1–4.
- [20] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *International Conference on VLSI Design*, 2011, pp. 346–351.
- [21] V. Mrazek, S. S. Sarwar, L. Sekanina *et al.*, "Design of power-efficient approximate multipliers for approximate artificial neural networks," in *International Conference on Computer-Aided Design*, 2016, pp. 1–7.
- [22] S. Rehman, W. Elharouni, M. Shafique *et al.*, "Architectural-space exploration of approximate multipliers," in *International Conference on Computer-Aided Design*, 2016, pp. 1–8.
- [23] H. Esmailzadeh, A. Sampson, L. Ceze *et al.*, "Neural acceleration for general-purpose approximate programs," in *International Symposium on Microarchitecture*, 2012, pp. 449–460.
- [24] Z. Du, A. Lingamneni, Y. Chen *et al.*, "Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators," in *Asia and South Pacific Design Automation Conference*, 2014, pp. 201–206.
- [25] S. Mazahir, O. Hasan, R. Hafiz *et al.*, "An area-efficient consolidated configurable error correction for approximate hardware accelerators," in *Design Automation Conference*, 2016, pp. 1–6.
- [26] S. Venkataramani, V. K. Chippa, S. Chakradhar *et al.*, "Quality programmable vector processors for approximate computing," in *International Symposium on Microarchitecture*, 2013, pp. 1–12.
- [27] L. Li and H. Zhou, "On error modeling and analysis of approximate adders," in *International Conference on Computer-Aided Design*, 2014, pp. 511–518.
- [28] 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.
- [29] M. K. Ayub, O. Hasan, and M. Shafique, "Statistical error analysis for low power approximate adders," in *Design Automation Conference*, 2017.
- [30] 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.
- [31] S. Mazahir, O. Hasan, R. Hafiz *et al.*, "Probabilistic error modeling for approximate adders," *IEEE Transactions on Computers*, vol. 66, no. 3, pp. 515–530, 2017.
- [32] T. H. Cormen, C. E. Leiserson, R. L. Rivest *et al.*, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009.
- [33] N. Zhu, W. L. Goh, G. Wang *et al.*, "Enhanced low-power high-speed adder for error-tolerant application," in *International SoC Design Conference*, 2010, pp. 323–327.

## APPENDIX A

### PROOF OF THEOREM 1

To prove the claim, we first introduce the following lemma.

#### Lemma 3

If the speculated carry-in of block  $j$ ,  $c_j^*$ , is 0 and the group propagate signals  $P_{i-1} = \dots = P_j = 1$ , where  $i > j$ , then the speculated carry-in of block  $i$ ,  $c_i^*$ , is 0.

*Proof:* We discuss based on the following two cases:

- 1) The case where  $cl_i \geq sl_j$ . For this case, carry generator  $CG_i$  ends before  $sl_j$ . Since  $P_{i-1} = \dots = P_j = 1$ , we have  $c_i^* = 0$ .
- 2) The case where  $cl_i < sl_j$ . We prove that  $c_i^* = 0$  by contradiction. Assume that  $c_i^* = 1$ . Since  $P_{i-1} = \dots = P_j = 1$ , we must have  $G_{(sl_j-1):cl_i} = 1$ . Given the assumption that  $cl_j \leq cl_i$ , we can see that the leading bits of carry generator  $CG_j$  produces a generate signal. Thus,  $c_j^* = 1$ , which contradicts the assumption.

For both cases, we conclude that  $c_i^* = 0$ .  $\square$

Now, we prove the claim. We first analyze the accurate and approximate sums. The accurate sum is

$$S = \sum_{i=0}^{m-1} 2^{sl_i} (A_i + B_i), \quad (20)$$

where  $A_i$  and  $B_i$  ( $0 \leq i \leq m-1$ ) represent the values encoded by block  $i$  of input  $A$  and block  $i$  of input  $B$ , respectively.

The approximate sum is

$$S^* = \sum_{i=0}^{m-1} 2^{sl_i} S_i^*, \quad (21)$$

where  $S_i^*$  ( $0 \leq i \leq m-1$ ) represents the value produced by sub-adder  $i$  of the approximate adder. For block  $i$  where  $0 \leq i < m-1$ ,  $S_i^*$  is only a  $k_i$ -bit binary number. If there is a carry-out produced by the sub-adder, it is dropped. Therefore, for  $0 \leq i < m-1$ ,

$$S_i^* = A_i + B_i + c_i^* - 2^{k_i} c_{out,i}^*, \quad (22)$$

where  $c_i^*$  is the carry-in to block  $i$  produced by carry generator  $CG_i$  and  $c_{out,i}^*$  is the carry-out of the sum  $A_i + B_i + c_i^*$ . For example, if  $k_i = 4$ ,  $A_i = (0000)_2$ ,  $B_i = (1111)_2$ , and  $c_i^* = 1$ , then we have  $c_{out,i}^* = 1$  and  $S_i^* = (0000)_2$ . For block  $(m-1)$ , since its sub-adder also produces the carry-out, we have

$$S_{m-1}^* = A_{m-1} + B_{m-1} + c_{m-1}^*. \quad (23)$$

Given Eqs. (21), (22) and (23) and the assumption that  $c_0^* = 0$ , we have

$$\begin{aligned} S^* &= \sum_{i=0}^{m-1} 2^{sl_i} S_i^* = (A_{m-1} + B_{m-1} + c_{m-1}^*) 2^{sl_{m-1}} \\ &+ \sum_{i=0}^{m-2} \left( A_i + B_i + c_i^* - 2^{k_i} c_{out,i}^* \right) 2^{sl_i} \\ &= \sum_{i=1}^{m-1} (A_i + B_i + c_i^* - c_{out,i-1}^*) 2^{sl_i} + (A_0 + B_0) \end{aligned} \quad (24)$$

Given Eqs. (20) and (24), we obtain

$$S - S^* = \sum_{i=1}^{m-1} (c_{out,i-1}^* - c_i^*) 2^{sl_i}. \quad (25)$$

For any  $1 \leq i \leq m-1$ , each input combination belongs to one of the following three cases:

- 1) The case where  $P_{i-1} = 0$ ;
- 2) The case where  $P_{i-1} = 1$  and  $c_{i-1}^* = c_{i-1}$ ;
- 3) The case where  $P_{i-1} = 1$  and  $c_{i-1}^* \neq c_{i-1}$ .

For case 1, we have either  $G_{i-1} = 0$  or  $K_{i-1} = 0$ . For both situations, the carry-out of block  $(i-1)$  does not depend on the carry-in to block  $(i-1)$ . Thus, the carry-out of the sum  $A_{i-1} + B_{i-1} + c_{i-1}^*$ ,  $c_{out,i-1}^*$ , is equal to the correct carry-in to block  $i$ ,  $c_i$ .

For case 2, since  $c_{i-1}^* = c_{i-1}$ , we have  $A_{i-1} + B_{i-1} + c_{i-1}^* = A_{i-1} + B_{i-1} + c_{i-1}$ . Note that  $c_{out,i-1}^*$  is the carry-out of the sum  $A_{i-1} + B_{i-1} + c_{i-1}^*$ , while  $c_i$  is the carry-out of the sum  $A_{i-1} + B_{i-1} + c_{i-1}$ . Therefore, we have  $c_{out,i-1}^* = c_i$ .

In summary, for both cases 1 and 2, we conclude that  $c_{out,i-1}^* - c_i^* = c_i - c_i^*$ . Since  $c_i^*$  is produced by the truncated carry chain, we always have  $c_i^* \leq c_i$ . Thus,  $c_{out,i-1}^* - c_i^* \geq 0$ . Furthermore,  $c_{out,i-1}^* - c_i^* = 1$  if and only if  $c_i = 1$  and  $c_i^* = 0$ .

For case 3, since  $c_{i-1}^* \neq c_{i-1}$ , the only situation is that  $c_{i-1}^* = 0$  and  $c_{i-1} = 1$ . The other situation where  $c_{i-1}^* = 1$  and  $c_{i-1} = 0$  cannot occur, since if  $c_{i-1} = 0$ , the speculated carry-in  $c_{i-1}^*$  produced by the truncated carry chain must also be 0. Now, given  $P_{i-1} = 1$  and  $c_{i-1}^* = 0$ , the carry-out of the sum  $A_{i-1} + B_{i-1} + c_{i-1}^*$  is 0, i.e.,  $c_{out,i-1}^* = 0$ . Furthermore, since  $c_{i-1}^* = 0$  and  $P_{i-1} = 1$ , by Lemma 3, we have  $c_i^* = 0$ . Therefore, for this case,  $c_{out,i-1}^* - c_i^* = 0$ .

Based on the above discussion, we can see that for any  $1 \leq i \leq m-1$ ,  $0 \leq c_{out,i-1}^* - c_i^* \leq 1$ . By Eq. (25), we have  $S - S^* \geq 0$ . The error distance  $|err| = |S^* - S| = S - S^*$ . Furthermore, the 1s in the error pattern can only occur at position  $sl_i$ , where  $1 \leq i \leq m-1$ . If there is a 1 at position  $sl_i$  of the error pattern, we must have  $c_{out,i-1}^* - c_i^* = 1$ . By the above discussion, this happens if and only if  $c_i = 1$ ,  $c_i^* = 0$ , and the condition that  $P_{i-1} = 1$  and  $c_{i-1}^* \neq c_{i-1}$  is not satisfied.

## APPENDIX B

### PROOF OF THEOREM 2

Since there is a 1 at position  $sl_i$  ( $1 \leq i \leq m-1$ ) of the error pattern, by Theorem 1, we have  $c_i = 1$  and  $c_i^* = 0$ .

We first prove that  $P_{i-1} = \dots = P_{i-t_i} = PL_i = 1$ , i.e., each bit covered by  $CG_i$  should have its propagate signal as 1. We prove this by contradiction. If there is at least one bit covered by  $CG_i$  that has its generate or kill signal as 1, we find the leftmost bit that satisfies this condition. For this bit, if its generate signal is 1, it will cause  $c_i = c_i^* = 1$ ; Otherwise, if its kill signal is 1, it will cause  $c_i = c_i^* = 0$ . Both cases will lead to a contradiction to the assumption that  $c_i = 1$  and  $c_i^* = 0$ . Therefore, we conclude that each bit covered by  $CG_i$  should have its propagate signal as 1.

We now prove that  $G_{(cl_i-1):0} = 1$ , i.e., the group generate signal of the remaining bits not covered by  $CG_i$  should be 1. This can also be proved by contradiction. If this is not true, then either the group propagate or the group kill signal of the remaining bits is 1. For both cases, given that  $P_{i-1} = \dots = P_{i-t_i} = PL_i = 1$ , we can deduce that  $c_i = c_i^* = 0$ , leading to a contradiction.

## APPENDIX C

### PROOF OF THEOREM 3

First, by Theorem 1, we have the following corollary.

### Corollary 2

If  $P_{i-1} = 1$  and  $c_{i-1}^* \neq c_{i-1}$ , where  $1 \leq i \leq m-1$ , then position  $sl_i$  of the error pattern cannot be 1.

Now, we prove the claim in the theorem. We study the condition for event  $E_{i,j}$  to occur. By the definition of  $E_{i,j}$ , there is a 1 at bit position  $sl_i$  of the error pattern. By Theorem 1, we have  $c_i = 1$  and  $c_i^* = 0$ . Since  $cl_i = cl_{i-1}$ , carry generator  $CG_i$  at least covers the entire block  $(i-1)$ . By Theorem 2, we have  $P_{i-1} = 1$ .

Given that  $cl_i = cl_{i-1}$ , carry generator  $CG_i$  covers the entire carry generator  $CG_{i-1}$ . Now, combining this fact with the claims  $c_i^* = 0$  and  $P_{i-1} = 1$ , we have  $c_{i-1}^* = 0$ . In addition, since  $P_{i-1} = 1$  and  $c_i = 1$ , we have the correct carry-in  $c_{i-1} = c_i = 1$ . Now, given that  $c_{i-1} = 1$ ,  $c_{i-1}^* = 0$ , and  $P_{i-1} = 1$ , by Corollary 2, the position  $sl_i$  of the error pattern cannot be 1. Thus, we conclude that when  $cl_i = cl_{i-1}$ , event  $E_{i,j}$  cannot occur and the probability  $e_{i,j} = 0$ .

## APPENDIX D

### PROOF OF THEOREM 4

We study the condition for event  $E_{i,j}$  to occur. Since  $j < i$  and  $i - j \leq t_i$ , we have  $t_i \geq 1$ . By the definition of  $E_{i,j}$ , there are 1s at bit positions  $sl_i$  and  $sl_j$  of the error pattern. Since bit position  $sl_i$  in the error pattern is 1, by Theorem 2,  $P_{i-1} = \dots = P_{i-t_i} = 1$ . Since  $i - j \leq t_i$ , we further have  $P_{i-1} = \dots = P_j = 1$ .

Since there is a 1 at position  $sl_j$  of the error pattern, by Theorem 1,  $c_j = 1$  and  $c_j^* = 0$ . Given  $c_j^* = 0$  and  $P_{i-2} = \dots = P_j = 1$ , by Lemma 3,  $c_{i-1}^* = 0$ . Furthermore, since  $c_j = 1$  and  $P_{i-2} = \dots = P_j = 1$ , the correct carry-in  $c_{i-1} = 1$ . Now, we have  $c_{i-1}^* = 0$ ,  $c_{i-1} = 1$ , and  $P_{i-1} = 1$ . By Corollary 2, position  $sl_i$  of the error pattern cannot be 1. Therefore, we conclude that when  $i - j \leq t_i$ , event  $E_{i,j}$  cannot occur and the probability  $e_{i,j} = 0$ .

## APPENDIX E

### PROOF OF LEMMA 1

By Theorems 1 and 2, in order to have a 1 at position  $sl_i$  of the error pattern, we must have  $c_i = 1$ ,  $c_i^* = 0$ , and  $P_{i-1} = \dots = P_{i-t_i} = PL_i = 1$ .

To prove the claim, we differentiate the approximate adder by whether  $t_i$  equals 0.

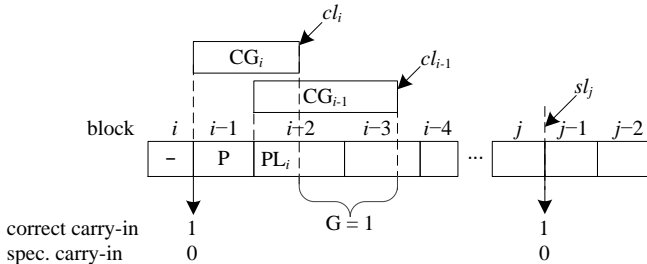


Fig. 8: An example approximate adder satisfying  $cl_{i-1} \geq sl_j$  and  $t_i > 0$ .

We first discuss the case where  $t_i > 0$ , which is illustrated in Fig. 8. In order to prove that  $G_{(cl_i-1):cl_{i-1}} = 1$ , we only need to show that neither  $K_{(cl_i-1):cl_{i-1}}$  nor  $P_{(cl_i-1):cl_{i-1}}$  can be 1.

We prove this by contradiction. If either  $K_{(cl_i-1):cl_{i-1}}$  or  $P_{(cl_i-1):cl_{i-1}}$  is 1, then due to  $P_{i-2} = \dots = P_{i-t_i} = PL_i = 1$ , we have the speculated carry-in of block  $(i-1)$ ,  $c_{i-1}^*$ , is 0.

Since  $c_i = 1$  and  $P_{i-1} = 1$ , we must have the correct carry-in  $c_{i-1} = c_i = 1$ . Now, given that  $P_{i-1} = 1$  and  $c_{i-1} \neq c_{i-1}^*$ , by Corollary 2, position  $sl_i$  of the error pattern cannot be 1. Thus, we must have  $G_{(cl_i-1):cl_{i-1}} = 1$ .

We now discuss the case where  $t_i = 0$ , which is illustrated in Fig. 9.

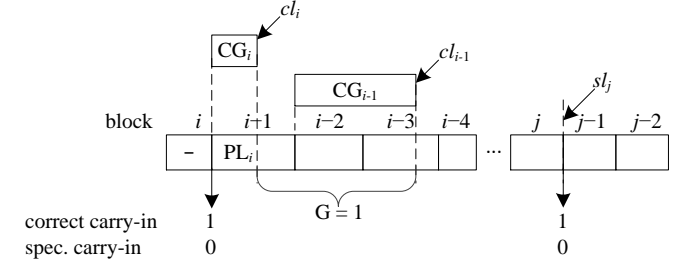


Fig. 9: An example approximate adder satisfying  $cl_{i-1} \geq sl_j$  and  $t_i = 0$ .

In this case, the set of bits from position  $(cl_i - 1)$  to  $cl_{i-1}$  can be partitioned into two subsets: the set of bits from position  $(cl_i - 1)$  to  $sl_{i-1}$  and the set of bits from position  $(sl_{i-1} - 1)$  to  $cl_{i-1}$ . Consider the bits from position  $(cl_i - 1)$  to  $sl_{i-1}$ . If their group kill signal is 1, then we obtain  $c_i = 0$ , which contradicts our previous claim that  $c_i = 1$ . If their group generate signal is 1, then we immediately have  $G_{(cl_i-1):cl_{i-1}} = 1$ .

Now, we consider the situation where the the group propagate signal of the bits from position  $(cl_i - 1)$  to  $sl_{i-1}$  is 1. For this case, given that  $PL_i = 1$ , we have  $P_{i-1} = 1$ . Under this condition, by a similar proof as the case where  $t_i > 0$ , we can show that neither  $K_{(sl_{i-1}-1):cl_{i-1}}$  nor  $P_{(sl_{i-1}-1):cl_{i-1}}$  can be 1. Thus,  $G_{(sl_{i-1}-1):cl_{i-1}}$  must be 1. Thus, we also have  $G_{(cl_i-1):cl_{i-1}} = 1$ .

In conclusion, for the case where  $t_i = 0$ , we also have  $G_{(cl_i-1):cl_{i-1}} = 1$ .

## APPENDIX F

### PROOF OF THEOREM 5

We consider the carry generator of block  $(r+j)$  ( $0 \leq r \leq i-j$ ) of the original approximate adder. There are two cases for it.

- 1) It does not cover any bits on the right of the lowest bit of block  $j$ . Then, the speculated carry-in to block  $(r+j)$  of the original approximate adder is equal to that to block  $r$  of the imaginary approximate adder.
- 2) It covers some bits on the right of the lowest bit of block  $j$ . Since there is a 1 at position  $sl_j$  of the error pattern of the original approximate adder, by Theorem 2, all the bits covered by the carry generator of block  $j$  have their propagate signals as 1. Due to the assumption that  $cl_t \geq cl_{t-1}$  for any  $0 < t < m$ , the bits in the carry generator of block  $(r+j)$  and on the right of the lowest bit of block  $j$  are at higher positions than  $cl_j$ . Therefore, these bits must have their propagate signals as 1. Given this, the speculated carry-in to position  $sl_j$  in the carry generator  $CG_{r+j}$  of the original approximate adder is 0, equal to the speculated carry-in to the carry generator  $CG_r$  of the imaginary approximate adder. Thus, the output of the carry generator  $CG_{r+j}$  of the original approximate adder is equal to that of the carry generator  $CG_r$  of the imaginary approximate adder. This means the speculated carry-in to block  $(r+j)$  of the original approximate adder is equal to that to block  $r$  of the imaginary approximate adder.



In conclusion, for any  $0 \leq r \leq i - j$ , the speculated carry-in to block  $(r + j)$  of the original approximate adder is equal to that to block  $r$  of the imaginary approximate adder. Thus, the sum produced by the imaginary approximate adder is equal to  $(s_{sl_{i+1}-1}^* \cdots s_{sl_j}^*)_2$ .