A General Sign Bit Error Correction Scheme for Approximate Adders

Rui Zhou and Weikang Qian
University of Michigan-Shanghai Jiao Tong University Joint Institute
Shanghai Jiao Tong University, Shanghai, China
Email: {zhouruisjtu, qianwk}@sjtu.edu.cn

ABSTRACT

Approximate computing is an emerging design technique for error-tolerant applications. As adders are the key building blocks in many applications, approximate adders have been widely studied recently. However, existing approximate adders may introduce sign bit error when doing two's complement signed addition, which is not tolerable for some applications. In this work, we propose a scheme that can correct sign bit error with low area and delay overhead. It is a general design applicable to many block-based approximate adders. This design not only can correct the sign bit error when it occurs, but also can fix some errors in the most significant bits even if there is no sign bit error. Experimental results on a real application, namely edge detection, showed that the approximate adders with our sign bit error correction module were up to 5.5 times better in peak signal-to-noise ratio than the original approximate adders, while the area and delay overhead is small.

Keywords

Approximate computing, approximate adder, sign bit error correction

1. INTRODUCTION

With the continuous scaling down of CMOS transistors, energy consumption has become a major concern in designing VLSI chips. Approximate computing, an emerging technology that improves circuit performance and energy efficiency by allowing a little amount of inaccuracy in the computation result, attracts many research efforts recently, due to its applicability to many error-tolerant applications, such as image processing, machine learning, multimedia, and data mining [1].

As adders are key building blocks for many error-tolerant applications, a number of approximate adders were proposed recently [2-9]. Most of them fall into the category of *block-based approximate adder*. It splits the entire adder into several blocks of sub-adders in order to reduce computation delay. Each sub-adder takes a speculated carry-in, which is produced based on some previous input bits. The computation of each sub-adder is correct. Due to the popularity of the block-based approximate adder, it is the focus of this work.

GLSVLSI'16, May 18-20, 2016, Boston, MA, USA.
 ISBN 978-1-4503-4274-2/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2902961.2903012

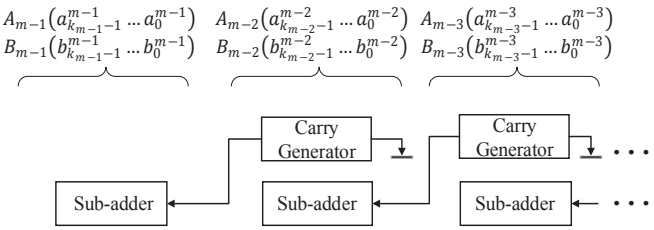


Fig. 1: Error Tolerant Adder Type II (ETAII) proposed in [2].

A representative of the block-based approximate adder is the Error Tolerant Adder II (ETAII) proposed in [2], which is shown in Fig. 1. In ETAII, it segments the whole adder into m blocks, and the carry-in signal of each sub-adder is produced by a carry generator operating on the input bits in the previous block. The carry-in signal of each carry generator is set as 0. There are many other examples of block-based approximate adder. The Carry Skip Approximate Adder (CSA) proposed in [3] also has m blocks, but the carry-in signal of the i-th sub-adder depends on input bits of the $(i-1)$-th block and the $(i-2)$-th block. The Low Relative Error Approximate Adder (LREA) proposed in [4] and the Speculative Carry Select Adder (SCSA) proposed in [5] have similar design styles.

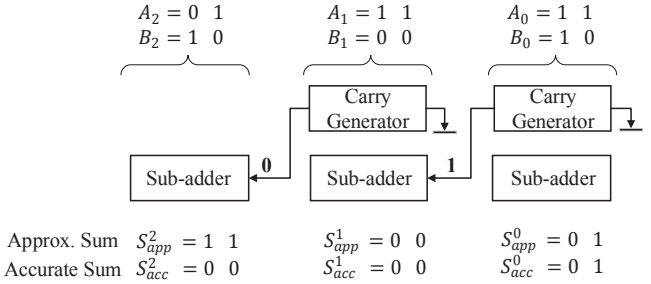


Fig. 2: An example of sign bit error in ETAII.

However, all these typical approximate adders mentioned above may generate an error in the sign bit when doing two's complement signed addition. For example, in a 6-bit 3-block ETAII adder, the sign bit is wrong when the two inputs are 011111 (base 2) and 100010 (base 2), as shown in Fig. 2. The approximate sum is 110001 (base 2), but the accurate sum is 000001 (base 2). The error is caused by the wrong carry-in signal fed into the leftmost sub-adder, which is produced by the carry generator in the previous block.

Having a sign bit error in approximate addition may cause a catastrophic failure for many applications and may not be tolerable. There are only few existing approximate adders which can correct the sign bit error, and their designs all have some limitations. For example, the Variable Latency Speculative Adder (VLSA) proposed in [6] combines a proposed approximate adder with an accurate adder for error recovery. However, it requires a large circuit area. To ensure correct sign bit calculation, it takes

two clock cycles. Furthermore, when the correction module is enabled, its result is always correct, which is not necessary for error-tolerant applications. In the LREA, the authors proposed a sign bit error correction module, but it is only applicable to LREA itself [4].

In this paper, we proposed a low-overhead module to correct sign bit error for block-based approximate adders. Compared to the previous approaches used in [4] and [6], our module is a general design which is applicable to many existing approximate adders regardless how the speculated carry-in to the sub-adder is generated. The only requirement is that all sub-adders are correct.

In summary, the main contributions of our work are as follows:

1) We give a general and comprehensive analysis on when sign bit error occurs.
2) Based on the error analysis, we propose an efficient design to correct the sign bit error.
3) We apply the proposed design to several existing approximate adders and experimentally demonstrate its good performance.

The remainder of this paper is organized as follows. In Section 2, we give an analysis on when sign bit error occurs. In Section 3, we propose a method to correct sign bit error. In Section 4, we show our optimized circuit design. In Section 5, we show the experimental results. Finally, in Section 6, we draw the conclusion.

2. ERROR ANALYSIS

In this section, we will analyze all kinds of situations where a sign bit error might occur in the computation of approximate adder and provide the correction mechanism for each situation. In our analysis, we assume that the inputs do not cause an overflow.

The only pre-requisite of our analysis is that the sub-adder of each block is correct. This requirement is satisfied for all the existing block-based approximate adders. Before the formal analysis, we define the following notations.

- $A = (a_{n-1}a_{n-2} \dots a_0)$ and $B = (b_{n-1}b_{n-2} \dots b_0)$ are the two inputs of the adder, where a_i and b_i represent the i-th bits of A and B, respectively
- We assume the approximate adder has m blocks and the i-th $(0 \le i \le m-1)$ block has k_i bits. Therefore, we have $n = \sum_{i=0}^{m-1} k_i$.
- s_i, c_i, p_i, and g_i represent the i-th $(0 \le i \le n-1)$ sum bit, carry-out bit, propagation bit, and generation bit of the accurate adder, respectively. c_{in} represents the carry-in of the entire adder. For the ease of discussion, we define $c_{-1} = c_{in}$. By the behavior of the adder, we have

$$s_i = a_i \oplus b_i \oplus c_{i-1}, \quad c_i = a_i b_i + (a_i + b_i)c_{i-1}$$
$$p_i = a_i \oplus b_i, \quad g_i = a_i b_i.$$

Note that the sum bit and the carry-out bit can be calculated by the propagate bit and the generate bit as follows:

$$s_i = p_i \oplus c_{i-1}, \quad c_i = g_i + p_i c_{i-1}.$$

For analysis purpose, we also split the accurate adder into m blocks. For each block of approximate adder and accurate adder, we define the following signals.

- $C^i_{app\_sum\_in}$: the carry-in signal of the i-th sub-adder of the approximate adder. This signal varies for different approximate adders.
- $C^i_{acc\_sum\_in}$: the carry-in signal of the i-th block of the accurate adder.
- S^i_{app}: the k_i-bit sum of the i-th sub-adder of the approximate adder.
- S^i_{acc}: the k_i-bit sum of the i-th block of the accurate adder.
- $s^i_{app,j}$ $(0 \le j \le k_i - 1)$: the sum bit at position j in block i of the approximate adder.
- $s^i_{acc,j}$ $(0 \le j \le k_i - 1)$: the sum bit at position j in block i of the accurate adder.
- $c^i_{app,j}$ $(0 \le j \le k_i - 2)$: the carry-out bit at position j in the i-th sub-adder of the approximate adder.
- $c^i_{acc,j}$ $(0 \le j \le k_i - 2)$: the carry-out bit at position j in block i of the accurate adder.
- A_i and B_i: the 2 k_i-bit inputs of block i.
- a^i_j and b^i_j $(0 \le j \le k_i - 1)$: the 2 input bits at position j in block i.
- p^i_j $(0 \le j \le k_i - 1)$: the propagate bit at position j in block i.
- g^i_j $(0 \le j \le k_i - 1)$: the generate bit at position j in block i.

Some of the above signals for approximate adder and accurate adder are shown in Fig. 3(a) and 3(b), respectively.

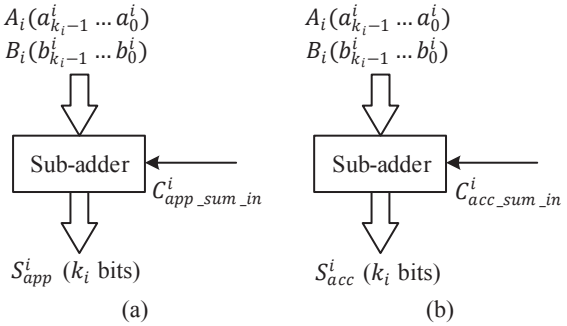


Fig. 3: Signal definitions for (a) an approximate adder and (b) an accurate adder.

The necessary condition for a sign bit error to occur is that for block $m-1$, $C^{m-1}_{app\_sum\_in} \neq C^{m-1}_{acc\_sum\_in}$. This condition can be further divided into two cases:

- Case 2.1: $C^{m-1}_{app\_sum\_in} = 0$ and $C^{m-1}_{acc\_sum\_in} = 1$;
- Case 2.2: $C^{m-1}_{app\_sum\_in} = 1$ and $C^{m-1}_{acc\_sum\_in} = 0$.

We will discuss them separately in the following sections.

2.1 $C^{m-1}_{app\_sum\_in} = 0$ and $C^{m-1}_{acc\_sum\_in} = 1$

Without loss of generality, we assume that the input $A \ge B$. Then, there are four cases where a sign bit error occurs:

1. $A \ge 0, B \ge 0, S^{m-1}_{app} < 0$;
2. $A < 0, B < 0, S^{m-1}_{app} \ge 0$;
3. $A \ge 0, B < 0, S^{m-1}_{acc} \ge 0, S^{m-1}_{app} < 0$;
4. $A \ge 0, B < 0, S^{m-1}_{acc} < 0, S^{m-1}_{app} \ge 0$.

We consider these four cases in the following subsections.

2.1.1 The Case where $A \ge 0, B \ge 0, S^{m-1}_{app} < 0$

This case is shown in Fig. 4(a). In this case, we must have $S^{m-1}_{app} = 11 \dots 11$ and $S^{m-1}_{acc} = 00 \dots 00$. However, since $C^{m-1}_{app\_sum\_in} = 0$ and $s^{m-1}_{app,j} = 1$ for $j = 0,1, \dots, k_{m-1} - 2$, we must have $a^{m-1}_j \oplus b^{m-1}_j = 1$ for $j = 0,1, \dots, k_{m-1} - 2$. Thus, the speculated carry-in to the sign bit is $c^{m-1}_{app,k_{m-1}-2} = 0$. Given that the sign bits of the inputs A and B are both 0, we must have $S^{m-1}_{app} = 01 \dots 11$, which contradicts with the claim that $S^{m-1}_{app} = 11 \dots 11$. Therefore, this situation cannot happen.

2.1.2 The Case where $A < 0, B < 0, S^{m-1}_{app} \ge 0$

This case is shown in Fig. 4(b). In this case, we must have $S^{m-1}_{app} = 01 \dots 11$ and $S^{m-1}_{acc} = 10 \dots 00$. This situation is possible. For example, when $A_{m-1} = 1010$ and $B_{m-1} = 1101$, then we have $S^{m-1}_{app} = 0111$ and $S^{m-1}_{acc} = 1000$. To correct the error in this case, we need to invert the sign bit and set the other sum bits in block $(m-1)$ to 0.

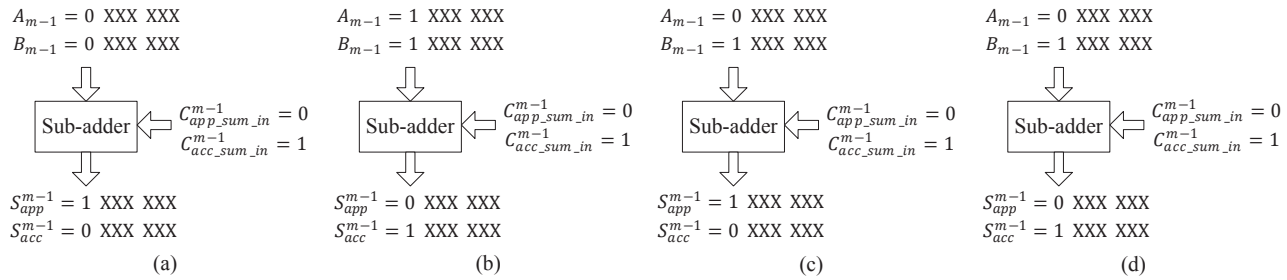


Fig. 4: Illustration of (a) Case 2.1.1, (b) Case 2.1.2, (c) Case 2.1.3, and (d) Case 2.1.4.

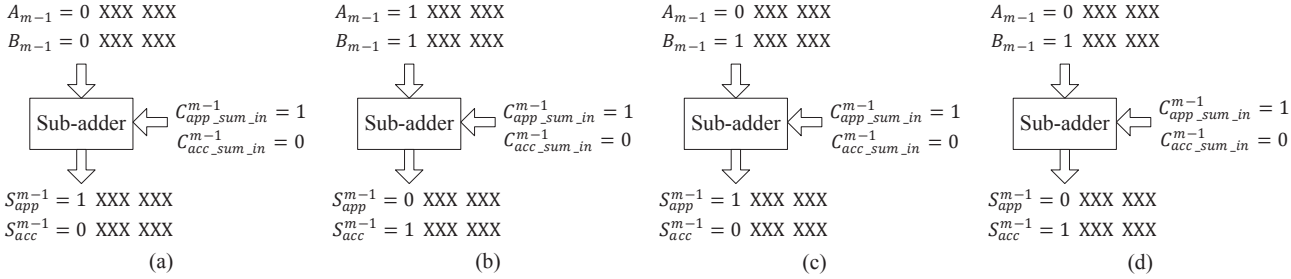


Fig. 5: Illustration of (a) Case 2.2.1, (b) Case 2.2.2, (c) Case 2.2.3, and (d) Case 2.2.4.

2.1.3 The Case where $A \geq 0, B < 0, S_{acc}^{m-1} \geq 0, S_{app}^{m-1} < 0$

This case is shown in Fig. 4(c). In this case, we must have $S_{app}^{m-1} = 11 \ldots 11$ and $S_{acc}^{m-1} = 00 \ldots 00$. This situation is possible. For example, when $A_{m-1} = 0010$ and $B_{m-1} = 1101$, then we have $S_{app}^{m-1} = 1111$ and $S_{acc}^{m-1} = 0000$. To correct the error in this case, we need to invert the sign bit and set the other sum bits in block $(m-1)$ to 0.

2.1.4 The Case where $A \geq 0, B < 0, S_{acc}^{m-1} < 0, S_{app}^{m-1} \geq 0$

This case is shown in Fig. 4(d). In this case, we must have $S_{app}^{m-1} = 01 \ldots 11$ and $S_{acc}^{m-1} = 10 \ldots 00$. However, applying the same argument used for Case 2.1.1, we can show that S_{app}^{m-1} should be $11 \ldots 11$. Therefore, this situation cannot happen.

2.2 $C_{app\_sum\_in}^{m-1} = 1$ and $C_{acc\_sum\_in}^{m-1} = 0$

We consider the same four cases as shown in Section 2.1.

2.2.1 The Case where $A \geq 0, B \geq 0, S_{app}^{m-1} < 0$

This case is shown in Fig. 5(a). In this case, we must have $S_{app}^{m-1} = 10 \ldots 00$ and $S_{acc}^{m-1} = 01 \ldots 11$. This situation is possible. For example, when $A_{m-1} = 0010$ and $B_{m-1} = 0101$, then we have $S_{app}^{m-1} = 1000$ and $S_{acc}^{m-1} = 0111$. To correct the error in this case, we need to invert the sign bit and set the other sum bits in block $(m-1)$ to 1.

2.2.2 The Case where $A < 0, B < 0, S_{app}^{m-1} \geq 0$

This case is shown in Fig. 5(b). In this case, we must have $S_{app}^{m-1} = 00 \ldots 00$ and $S_{acc}^{m-1} = 11 \ldots 11$. However, since $C_{acc\_sum\_in}^{m-1} = 0$ and $s_{acc,j}^{m-1} = 1$ for $j = 0,1, \ldots, k_{m-1} - 2$, we must have $a_j^{m-1} \oplus b_j^{m-1} = 1$ for $j = 0,1, \ldots, k_{m-1} - 2$. Thus, the correct carry-in to the sign bit is $c_{acc,k_{m-1}-2}^{m-1} = 0$. Given that the sign bits of the inputs A and B are both 1, we must have $S_{acc}^{m-1} = 01 \ldots 11$, which contradicts with the claim that $S_{acc}^{m-1} = 11 \ldots 11$. Therefore, this situation cannot happen.

2.2.3 The Case where $A \geq 0, B < 0, S_{acc}^{m-1} \geq 0, S_{app}^{m-1} < 0$

This case is shown in Fig. 5(c). In this case, we must have $S_{app}^{m-1} = 10 \ldots 00$ and $S_{acc}^{m-1} = 01 \ldots 11$. However, applying the same argument used for Case 2.2.2, we can show that S_{acc}^{m-1} should be $11 \ldots 11$. Therefore, this situation cannot happen.

2.2.4 The Case where $A \geq 0, B < 0, S_{acc}^{m-1} < 0, S_{app}^{m-1} \geq 0$

This case is shown in Fig. 5(d). In this case, we must have $S_{app}^{m-1} = 00 \ldots 00$ and $S_{acc}^{m-1} = 11 \ldots 11$. This situation is possible. For example, when $A_{m-1} = 0010$ and $B_{m-1} = 1101$, then we have $S_{app}^{m-1} = 0000$ and $S_{acc}^{m-1} = 1111$. To correct the error in this case, we need to invert the sign bit and set the other sum bits in block $(m-1)$ to 1.

2.3 Summary

The above discussion considers all cases where a sign bit error might occur when we do two's complement addition. To sum up, if a sign bit error happens, we need to identify whether the case is Case 2.1 or Case 2.2. To correct a sign bit error for Case 2.1, we invert the sign bit and set all the other sum bits in block $(m-1)$ to 0. To correct a sign bit error for Case 2.2, we invert the sign bit and set all the other sum bits in block $(m-1)$ to 1.

3. METHODOLOGY FOR CORRECTING SIGN BIT ERROR

In this section, we show a general methodology for correcting sign bit error.

First, we define the group propagate signal pp_i for each block. For any block $0 \leq i \leq m-2$, $pp_i = \prod_{j=0}^{k_i-1} p_j^i$. If $pp_i = 1$, then all the propagate signals of block i are 1. Thus, in the accurate adder, the carry-in of block i propagates to the carry-out of block i, which is just the carry-in of block $(i+1)$. Therefore, we have $C_{acc\_sum\_in}^{i+1} = C_{acc\_sum\_in}^{i}$ when $pp_i = 1$.

The group propagate signal for block $(m-1)$ is defined slightly different from the other blocks in that it does not include

$p^{m-1}_{k_{m-1}-1}$. Specifically, $pp_{m-1} = \prod_{j=0}^{k_{m-1}-2} p_j^{m-1}$. For the ease of discussion, we also define $pp_{-1} = 0$.

When $pp_{m-1} = 0$, the carry-out signal $c^{m-1}_{app,k_{m-1}-2}$ in the $(m-1)$-th sub-adder of the approximate adder does not depend on the speculated carry-in signal of the sub-adder. Therefore, it is equal to the correct carry-out signal $c^{m-1}_{acc,k_{m-1}-2}$. As a result, the sign bit produced by the approximate adder is correct. Thus, in the following, we assume that $pp_{m-1} = 1$.

From the analysis shown in Section 2, we can see that to check whether there is a sign bit error, we need to know $C^{m-1}_{acc\_sum\_in}$.

To obtain $C^{m-1}_{acc\_sum\_in}$, we first modify each sub-adder in the approximate adder by adding some extra logic to produce a carry-out. The carry-out of the i-th sub-adder, denoted as $C^i_{app\_out}$, can be calculated by the existing signals in the sub-adder as

$$C^i_{app\_out} = g^i_{k_i-1} + p^i_{k_i-1} \cdot c^i_{app,k_i-2}.$$

The circuit is shown in Fig. 6. For the ease of discussion, we define $C^{-1}_{app\_out} = c_{in}$.

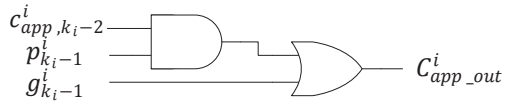


Fig. 6: Circuits for generating the carry-out signal of a sub-adder.

It is easy to see that when $pp_i = 0$, the carry-out $C^i_{app\_out}$ is equal to the correct carry-out in the accurate adder. Indeed, in our proposed design, the modification to each sub-adder is not necessary, as long as when $pp_i = 0$, there is a signal equal to the correct carry-out of block i. This is true for many existing approximate adders in which for any block $0 \le i \le m-2$, there is a carry generator that produces a speculated carry-out signal based on at least all the input bits in that block. For example, the ETAII shown in Fig. 1, the CSA proposed in [3], and the LREA proposed in [4] all satisfy this condition. For this type of approximate adder, when $pp_i = 0$, the speculated carry-out produced by the carry generator is equal to the correct carry-out of block i. We can just use the speculated carry-out as the signal $C^i_{app\_out}$, which takes no extra circuitry.

Since we assume $pp_{m-1} = 1$ and $pp_{-1} = 0$, there must exist a $0 \le t \le m-1$ such that $pp_{m-1} = pp_{m-2} = \cdots = pp_t = 1$ and $pp_{t-1} = 0$. This value t plays an important role in our correction scheme. Since $pp_{t-1} = 0$, by the above discussion, the signal $C^{t-1}_{app\_out}$ in the approximate adder gives the correct carry-out of block $t-1$, which is just the correct carry-in of block t, $C^t_{acc\_sum\_in}$. Because $pp_{m-2} = pp_{m-3} = \cdots = pp_t = 1$, the accurate carry-in of block $m-1$, $C^{m-1}_{acc\_sum\_in}$ is equal to $C^t_{acc\_sum\_in}$, which in turn is equal to $C^{t-1}_{app\_out}$. This gives us a way to obtain $C^{m-1}_{acc\_sum\_in}$. Note that the speculated carry-in $C^{m-1}_{app\_sum\_in}$ is also available in the given approximate adder. Then we can compare these two signals to check for sign bit error. If $C^{m-1}_{acc\_sum\_in} = C^{m-1}_{app\_sum\_in}$, there is no sign bit error. Otherwise, the computation might have a sign bit error and we can fix the error based on the different correction strategies discussed in Section 2.

However, in our actual implementation, we use a better strategy that not only can fix sign bit error when it occurs, but also can fix some errors in the most significant bits even if there is no sign bit error.

Our idea utilizes the signal $C^{t-1}_{app\_out}$ and the fact that $pp_{m-1} = \cdots = pp_t = 1$. When $C^{t-1}_{app\_out} = 0$, for the accurate adder, we have $s^i_{acc,j} = 1$ for $t \le i \le m-2$ and $0 \le j \le k_i - 1$, $s^{m-1}_{acc,j} = 1$ for $0 \le j \le k_{m-1} - 2$, and $c^{m-1}_{acc,k_{m-1}-2} = 0$. When $C^{t-1}_{app\_out} = 1$, for the accurate adder, we have $s^i_{acc,j} = 0$ for $t \le i \le m-2$ and $0 \le j \le k_i - 1$, $s^{m-1}_{acc,j} = 0$ for $0 \le j \le k_{m-1} - 2$, and $c^{m-1}_{acc,k_{m-1}-2} = 1$.

Summarizing the above patterns, we can conclude that no matter whether $C^{t-1}_{app\_out}$ is 0 or 1, for the accurate adder, we have $s^i_{acc,j} = \overline{C^{t-1}_{app\_out}}$ for $t \le i \le m-2$ and $0 \le j \le k_i - 1$, $s^{m-1}_{acc,j} = \overline{C^{t-1}_{app\_out}}$ for $0 \le j \le k_{m-1} - 2$, and $c^{m-1}_{acc,k_{m-1}-2} = C^{t-1}_{app\_out}$. Thus, no matter whether the above output bits in the approximate adder have error or not, we can simply set them to their correct values shown above. Once we get the correct carry-out $c^{m-1}_{acc,k_{m-1}-2}$, the correct sign bit can be easily obtained as

$$s^{m-1}_{acc,k_{m-1}-1} = p^{m-1}_{k_{m-1}-1} \oplus c^{m-1}_{acc,k_{m-1}-2}.$$

For the remaining sum bits in blocks $t-1, \ldots, 0$, we just keep them unchanged.

4. CIRCUIT DESIGN

In this section, we show the circuit design for correcting the sign bit error, which implements the methodology discussed in Section 3.

The implementation needs to address two key questions: (1) how to identify the specific value t for any input vector? (2) how to obtain the value $C^{t-1}_{app\_out}$?

To address the above questions, we define the following signals.

- $pc_i = pp_{m-1} \cdot pp_{m-2} \cdot \cdots \cdot pp_i$, for $0 \le i \le m-1$. By the definition of the value t, for $t \le i \le m-1$, we have $pc_i = 1$. For $0 \le i < t$, since $pp_{t-1} = 0$, we have $pc_i = 0$. This gives us a way to test whether $i \ge t$ or $i < t$.
- $pt_i = pc_i \cdot \overline{pp_{i-1}}$, for $0 \le i \le m-1$. By the definition of the value t, we have $pt_t = 1$. For any $t < i \le m-1$, since $pp_{i-1} = 1$, we have $pt_i = 0$. For any $0 \le i < t$, since $pc_i = 0$, we have $pt_i = 0$. In summary, among all pt_i's, only $pt_t = 1$. This gives us a way to identify t.
- $cv = \sum_{i=0}^{m-1} pt_i \cdot C^{i-1}_{app\_out}$. Since among all pt_i's, only $pt_t = 1$, we have $cv = C^{t-1}_{app\_out}$. This gives us a way to obtain $C^{t-1}_{app\_out}$.

The circuits for the signals pc_i, pt_i, and cv are shown in Fig. 7.

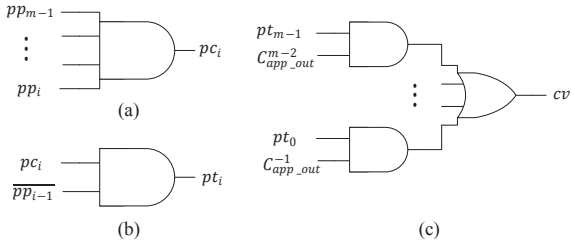


Fig. 7: Circuits for (a) the signal pc_i, (b) the signal pt_i, and (c) the signal cv.

For any $0 \le i \le m-2$, if $pc_i = 1$, then by the property of signal pc_i, we must have $i \ge t$. Based on the correction methodology discussed in Section 3, for any $0 \le j \le k_i - 1$, we set $s^i_{new,j} = \overline{cv}$, where $s^i_{new,j}$ is the new sum bit at position j in block i of the approximate adder. For any $0 \le i \le m-2$, if $pc_i = 0$, then we must have $i < t$. Based on the correction methodology, we keep $s^i_{new,j}$ as the old approximate value $s^i_{app,j}$. If $pc_{m-1} = 1$, then we set $s^{m-1}_{new,j} = \overline{cv}$, for any $0 \le j \le k_{m-1} - 2$,

and set $c^{m-1}_{new,k_{m-1}-2} = cv$, where $c^{m-1}_{new,k_{m-1}-2}$ is the new carry-out bit at position $k_{m-1}-2$ in block $m-1$ of the approximate adder. If $pc_{m-1} = 0$, the old sum bits $s^{m-1}_{app,j}$ $(0 \le j \le k_{m-1}-2)$ and the old carry-out bit $c^{m-1}_{app,k_{m-1}-2}$ are kept.

Based on the above discussion, we can realize $s^i_{new,j}$ by a MUX, as shown in Fig. 8(a). The new sign bit $s^{m-1}_{new,k_{m-1}-1}$ is realized by the circuit shown in Fig. 8(b).

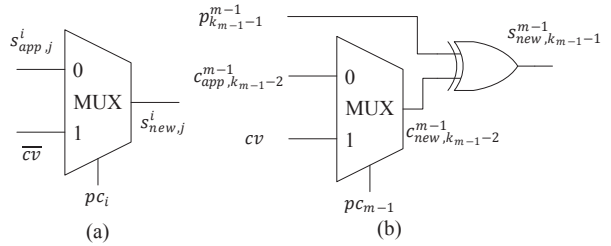


Fig. 8: Circuits for (a) the new sum bits and (b) the new sign bit of the approximate adder.

5. EXPERIMENTAL RESULTS

In this section, we present the experimental results on our proposed sign bit error correction module. The circuits were synthesized using Synopsys Design Compiler [10] and mapped to a 45nm Nangate cell library [11].

5.1 Overhead for Different Approximate Adders

We applied the proposed sign bit error correction module to three existing approximate adders, ETAII [2], CSA [3], and LREA [4]. Table 1 shows the area, delay, power consumption, and power-delay product (PDP) of the three approximate adders with (denoted by "sign") and without (denoted by "basic") the sign bit error correction module. For comparison purpose, we also list the four metrics of the conventional ripple carry adder (RCA) and carry look-ahead adder (CLA). All the adders are 16-bit adders. The three approximate adders are divided into 4 blocks of equal size of 4. For each approximate adder with the sign bit error correction module, the percentage increase of each metric due to the sign bit error correction module is shown inside the parentheses after the absolute value.

Table 1. Area, delay, power consumption, and power-delay product (PDP) of different 16-bit adders.

	Area (μm^2)	Delay (ns)	Power (mW)	PDP (fJ)
ETAII(basic)	99.5	0.97	22.3	21.6
ETAII (sign)	129.1 (30)	1.12 (15)	22.8 (2.2)	25.5 (18)
CSA (basic)	114.6	1.16	23.0	26.7
CSA (sign)	138.1 (21)	1.31 (13)	23.9 (3.9)	31.3 (17)
LREA(basic)	121.0	1.24	25.8	31.9
LREA (sign)	139.6 (15)	1.28 (3.2)	26.2 (1.6)	33.5 (5.0)
RCA	109.8	2.32	23.9	55.4
CLA	145.3	1.78	29.9	53.2

From the table, we can see that by adding the sign bit error correction module, the area increases 15%~30%, the delay increases 3.2%~15%, the power consumption increases 1.6%~3.9% and the PDP increases 5.0%~18%. The overhead for ETAII is the largest, since ETAII design is the simplest among the three. Overall, the overhead of the sign bit error correction module is small. Compared with RCA, although the approximate adders with the sign bit error correction module have slightly larger power consumption, they have much smaller delay and PDP. Compared with CLA, all of the four metrics of the approximate adders with the sign bit error correction module are smaller.

5.2 Overhead for Same Approximate Adder of Different Sizes

In this experiment, we applied the proposed sign bit error correction module to ETAIIs of different design parameters. We considered 5 different ETAIIs, which are listed in Table 2. The pair (n, k) indicates the adder is an n-bit adder with n/k blocks of equal size k. For example, the pair (32, 4) indicates a 32-bit ETAII with 8 blocks of equal size 4. Table 2 shows the area, delay, and power consumption of them with (column "sign") and without (column "basic") the sign bit error correction module. The values in the parentheses are the percentage increases.

Table 2. Area, delay, and power consumption of different ETAIIs with and without the sign bit error correction module.

ETAII parameters	Area (μm^2)		Delay (ns)		Power (mW)	
	basic	sign	basic	sign	basic	sign
(16, 2)	91.0	112.4 (24)	0.68	1.05 (54)	22.3	22.4 (0.4)
(16, 4)	99.5	129.1 (30)	0.97	1.12 (15)	22.3	22.8 (2.2)
(32, 2)	178.8	255.3 (43)	0.57	1.76 (209)	23.2	23.8 (2.6)
(32, 4)	208.5	261.8 (26)	0.97	1.42 (46)	23.6	23.8 (0.8)
(32, 8)	223.4	257.8 (15)	1.64	1.76 (7.3)	25.9	26.3 (1.5)

From the table, we can see that when the number of blocks increases, the overhead in area and delay due to the sign bit error correction module also increases. The reason is that some sub-modules in the sign bit error correction module, such as the circuits for the signals pc_i and cv, have area and delay complexity proportional to the number of blocks.

5.3 Performance Study on Edge Detection Application

To evaluate the effect of our sign bit error correction module in real applications, we applied several approximate adders with and without the sign bit error correction module to an image processing application, namely Roberts cross-based edge detection [12]. We used peak signal-to-noise ratio (PSNR) of the image as the performance metric. It is defined as

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right),$$

where MAX_I is the maximum possible pixel value of the image. MSE is the mean square error (MSE), which is defined as

$$MSE = \frac{1}{WH}\sum_{i=0}^{W-1}\sum_{j=0}^{H-1}[I(i,j) - K(i,j)]^2,$$

where W and H are the width and the height of an image, respectively. The values $I(i,j)$ and $K(i,j)$ are the results calculated by the accurate adder and the approximate adder, respectively, at location (i,j) in the image. For an accurate adder, its $MSE = 0$ and hence, its $PSNR = +\infty$. The higher the PSNR is, the better the performance of the adder is.

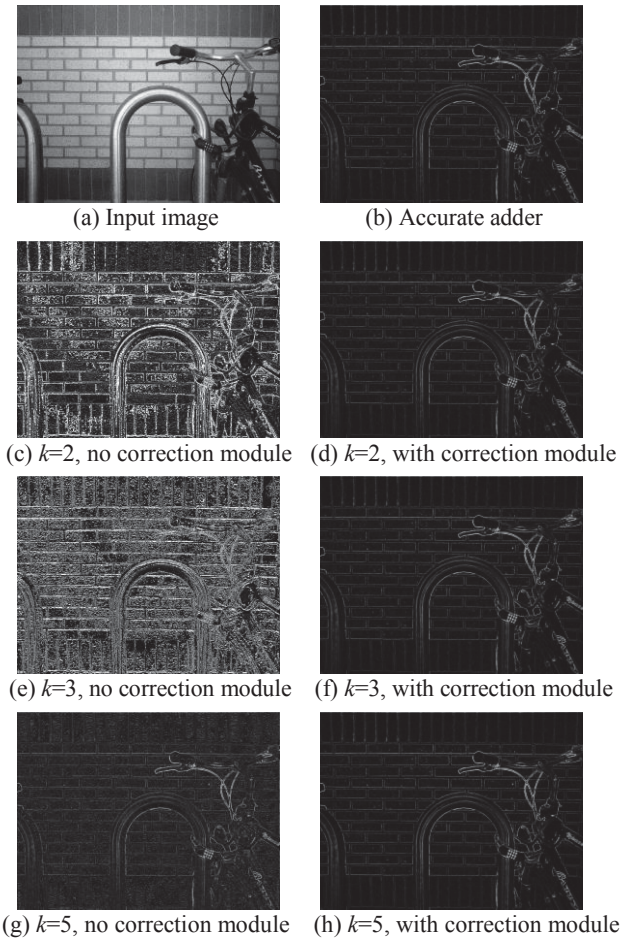
We used three existing approximate adders, ETAII, CSA, and LREA in our experiment. We applied these adders with and without the sign bit error correction module to perform edge detection on 10 sample images chosen from an online library [13].

Table 3 shows the average PSNR over the 10 images in our simulation. The adder parameter pairs (10, 2) and (10, 5) in the table have the same meaning as before. The pair (10, 3) refers to an adder with 4 blocks, of which the leftmost three are of size 3 and the rightmost one is of size 1. Note that some entries in the table is $+\infty$ because the MSE is either zero or very small. From the table, we can see that the approximate adders with the error correction module are up to 5.5 × better in PSNR than the original ones.

Table 3. PSNRs of different approximate adders with and without the sign bit error correction module.

Adder parameter	ETAII		CSA		LREA	
	basic	sign	basic	sign	basic	sign
(10, 2)	9.34	28.2	7.95	33.8	9.88	42.7
(10, 3)	11.0	31.7	13.3	42.6	9.83	54.5
(10, 5)	$+\infty$	$+\infty$	$+\infty$	$+\infty$	17.8	$+\infty$

To visually demonstrate the effectiveness of our proposed module, we show one sample image processed by the accurate adder and the LREAs with and without the sign bit error correction module for $n = 10$ and $k = 2,3,5$ in Fig. 9.



(a) Input image (b) Accurate adder

(c) *k*=2, no correction module (d) *k*=2, with correction module

(e) *k*=3, no correction module (f) *k*=3, with correction module

(g) *k*=5, no correction module (h) *k*=5, with correction module

Fig. 9: Roberts cross-based edge detection using different adders.

From Fig. 9, we can clearly see that the approximate adders with our sign bit error correction module have much better performance than those adders without the correction module.

Finally, we did another comparison between two approximate adders. One is an LREA with $(n, k) = (10,5)$ and no sign bit error correction module. The other is an LREA with $(n, k) = (10,2)$ and the sign bit error correction module. The comparison of their PSNRs, areas, delays, and power consumptions is shown in Table 4.

Table 4. Comparison of two approximate adders.

	PSNR	Area (μm^2)	Delay (ns)	Power (mW)
LREA (10, 5), basic	17.8	76.3	1.40	19.0
LREA (10, 2), sign	42.7	85.3	1.01	18.2

From the table, we can see that the LREA with $(n, k) = (10,2)$ and the sign bit error correction module is 12% larger in area than the LREA with $(n, k) = (10,5)$ and no sign bit error correction module. However, the delay and the power consumption of the former are 28% and 4.2% smaller than those of the latter, respectively. Furthermore, the PSNR of the former is 140% better than that of the latter. This result shows that in the case where the circuit area, delay, and power consumption are comparable, an approximate adder with the proposed sign bit error correction module is much better in PSNR than an approximate adder without the correction module.

6. CONCLUSION

In this paper, we proposed a general sign bit error correction scheme, which is applicable to many block-based approximate adders. The only requirement our scheme relies on is that each sub-adder in the approximate adder is correct, which is usually held. Our design is very efficient with low area, delay, and energy overhead. According to the properties of each specific approximate adder, the proposed design may be further simplified. The proposed module not only can correct the sign bit error when it occurs, but also can fix some errors in the most significant bits even if there is no sign bit error. Thus, it can further reduce the relative error and error rate of the original approximate adder. We applied our proposed module to an edge detection application and demonstrated the good performance of it.

ACKNOWLEDGEMENTS

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

7. REFERENCES

[1] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in ETS, 2013, pp. 1–6.
[2] N. Zhu, W. L. Goh, and K. S. Yeo, "An enhanced low-power high speed adder for error-tolerant application," in ISIC, 2009, pp. 69–72.
[3] Y. Kim, Y. Zhang, and P. Li, "An energy efficient approximate adder with carry skip for error resilient neuromorphic VLSI systems," in ICCAD, 2013, pp. 130–137.
[4] J. Hu and W. Qian, "A new approximate adder with low relative error and correct sign calculation," in DATE, 2015, pp. 1449–1454.
[5] K. Du, P. Varman, and K. Mohanram, "High performance reliable variable latency carry select addition," in DATE, 2012, pp. 1257–1262.
[6] A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: A new paradigm for arithmetic circuit design," in DATE, 2008, pp. 1250–1255.
[7] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in DAC, 2012, pp. 820–825.
[8] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in ICCAD, 2013, pp. 48–54.
[9] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient VLSI implementation of softcomputing applications," IEEE Transactions on Circuits and Systems I, vol. 57, no. 4, pp. 850–862, 2010.
[10] Design Compiler, Synopsys Inc., http://www.synopsys.com
[11] Nangate 45nm Library, Nangate Inc., http://www.nangate.com
[12] L. G. Roberts, "Machine perception of three-dimensional soups," Ph.D. dissertation, Massachusetts Institute of Technology, 1963.
[13] Image databases: standard test images, http://www.imageprocessingplace.com/root_files_V3/image_databases.htm