JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 36, 1-12 (2020)
DOI: 10.6688/JISE.202001_36(1).0001

An Asynchronous High-Performance Approximate Adder with Low-Cost Error Correction

TARAK K. KODALI[1], YILIN ZHANG[2], EUGENE JOHN[1] AND WEI-MING LIN[1]
[1]*Department of Electrical and Computer Engineering*
University of Texas at San Antonio
San Antonio, TX 78249 USA
[2]*Advanced Micro Devices, Inc.*
Austin, TX 78735 USA

Integer addition, being the fundamental process used in all other integer and floating point arithmetic operations, is the most important process used in computer systems and related applications. Many adder designs have been proposed in order to enhance the overall system performance. Approximation/speculation proves to be a very effective approach in leading to results faster than non-approximation techniques, though with its intrinsic drawback in having a potentially incorrect result. This paper proposes an approximate adder design with a very cost-effective error correction capability incorporated. In addition, with a built-in completion-detection mechanism, the proposed design is suitable for an asynchronous or variable-latency processing environment, and can deliver an expected completion time much shorter than all well-known parallel adders.

Keywords: approximate adder, speculative adder, speculative circuit, asynchronous adder, variable-latency adder

1. INTRODUCTION

Parallel (or semi-parallel) adders have been dominating all various applications where adder speed is a critical concern. These include Carry-Lookahead Adder, Kogge-Stone adder [1], Brent-Kung adder [2], and many of their variations spreading over the design spectrum of trade-off between cost and speed. All these adder designs always produce correct results but most without a completion-detection mechanism. Thus, they do not take advantage of cases when the finalized correct result is available earlier than the worst-case delay. Therefore, these adders are not in general suitable for systems that employ asynchronous processing for earlier event triggering or systems that can detect operation completion in a variant number of clock cycles.

Approximate computing or imprecise hardware have become very popular in the past decade due to the growing concern in speed and power consumption. Without having to guarantee always obtaining a correct result, especially in the areas of digital signal processing for image, speech, and video, artificial intelligence and machine learning, these circuits can afford using a smaller amount of hardware and/or reach an acceptable result faster [3-6].

Due to reasons aforementioned, approximate or speculative adder designs are among the most focused and have led to many breakthroughs by using a reduced number of transistors and by truncating the carry propagation chain for a speculation-based operation [7, 8]. Additional circuitries have been also established to provide error detection and

Received January 9, 2018; revised July 19, 2018; accepted July 26, 2018.
Communicated by Meng Chang Chen.

correction in [9-14]. Configurable-accuracy adders to reduce the delay of the traditional adder are proposed in [15, 16] by splitting the traditional adder into several overlapped sub-adders to obtain the approximate results. Obvious tradeoffs from these designs are in the additional cost and delay required for their error detection and correction circuitries. Several papers attempted to derive the error probability of their designs but most either rely on approximate derivation, or simply reach an incorrect conclusion due to some erroneous assumptions. Several other recent related results, such as [17] which further attempted to improve hardware utilization efficiency, or [18] in which error resilience is improved, and [19] in which error probability is reduced using a hierarchical design approach. A detailed probabilistic model for analyzing error occurrence is given in [20]. A design by separating carry generator and sum generator to reduce power consumption is proposed in [21].

This paper proposes an approximate adder design based on a simple bit-overlapping design in [8], and incorporates error-detection and error-correction mechanisms into the design to always ensure the final result is correct. When an error is detected, the error-correction mechanism is then triggered to correct the result. The almost negligible logic required for this add-on correction mechanism is just a very small fraction of the original approximating logic, and is much simpler than all the known approximate adder designs. An even simpler completion-detection mechanism is also built in for this design to take advantage of an asynchronous processing environment. In addition, a comprehensive and robust analytical derivation is given in this paper to deliver the correct error probability of the approximate adder without resorting to approximation or assuming incorrect mutual exclusiveness like others. We also show that the expected processing latency is much shorter than all the well-known parallel fast adders, while at the same time incurring a smaller amount of logic and power consumption.

2. APPROXIMATE ADDERS

2.1 The Lu Approximate Adder

The proposed approximate adder design is based on a well-known approximate adder, the Lu Approximate Adder (LAA) [8]. The fundamental concept behind this design is to generate an "approximate (speculative) carry" for each bit for deriving the final sum bits. Generating an always correct carry bit can only be guaranteed by using input operand bits from all less significant bit positions, which involves a potentially long chain of carry propagation. Instead, in this design each approximate carry bit is generated using only a fraction of input operands from the less significant portion. Since most of the actual carry-propagation chains are shorter than the worst case, such an approximate carry generated may be mostly correct, if not all.

For an n-bit addition to add two operands $A = a_{n-1}a_{n-2}\ldots a_1a_0$ and $B = b_{n-1}b_{n-2}\ldots b_1b_0$, instead of generating all correct carry bits $c_n, c_{n-1}, \ldots, c_2, c_1$, each approximate carry, denoted as ac_i for bit i, will be generated using a k-bit "Approximate Carry Generating Block" (ACGB), where $k < n$, with the following dependency: $c_i = ac_i$ if $i < k$. One can choose to use a simple k-bit Ripple Carry Adder (RCA), with a zero carry-in, to implement each such ACGB; however, the slow carry propagation of RCA, especially when k is not small, can easily beat the purpose of this design. Thus a more reasonable approach is to use the carry-lookahead circuit to produce each such speculative carry bit.

Adopting the general notations in the literature, let g_i and p_i denote the "carry-

generate" and "carry-propagate" signal for bit i, respectively, where $g_i = a_i \cdot b_i$ and $p_i = a_i \oplus b_i$. Thus, each ac_i can be generated with its corresponding k-bit ACGB as:

$$\begin{aligned} ac_i &= g_{i-1} + g_{i-2} \cdot p_{i-1} + g_{i-3} \cdot p_{i-1} \cdot p_{i-2} + \cdots \\ &\quad + g_{i-k} \cdot p_{i-1} \cdot p_{i-2} \cdots p_{i-k+1} \end{aligned} \tag{1}$$

where the summation and product notations are borrowed for the corresponding logical OR and AND operations, respectively. For the sake of generality and future extension, identical PGG circuits are used for all bit positions although the least significant one does not need to produce its p signal since p_{i-k} is not needed.

Note that obviously the carry bit thus generated is erroneous when $ac_i = 0$ but the actual carry c_i is 1, which arises under the following condition:

$$c_{i-k} = 1 \text{ and } \forall j, i-k \leq j \leq i-1, g_j = 0 \text{ and } p_j = 1.$$

That is, the length of carry propagation chain to lead to $c_i = 1$ is longer than k, which cannot be materialized by any m-bit ACGB with $m \leq k$. Let the n-bit LAA design using k-bit ACGBs be referred to as (n,k)-LAA.

2.2 Error Probability Analysis

The error probability analysis in [8] is not correctly presented, which is to be properly established here, since the error rate matters significantly in terms of performance for our proposed design and other similar designs.

First, as aforementioned, each of the ACGBs that are used to produce the carries at bit position i when $i \leq k$ will correctly generate their carry results since there will be no carry propagation chain longer than k for any of them. Thus, in a LAA all carry bits are considered correct from c_1 to c_3. For any other bit position i such that $i > k$, the approximate carry ac_i generated by its corresponding ACGB is correct only when there does not exist a carry propagation chain longer than k leading to its output carry.

Given any random input, the probability for any bit to have $p = 1$ is $1/2$ and to have $g = 1$ is $1/4$. Thus, for the example an LAA with $k = 3$, ac_5 is incorrect with the probability of $(\frac{1}{2})^3 \cdot \frac{1}{4} + (\frac{1}{2})^3 \cdot \frac{1}{2} \cdot \frac{1}{4}$, with the first term from a carry chain of 4 bits (from bit 1 to bit 4) and the second from a carry chain of 5 bits (from bit 0 to bit 4). In order to properly calculate the overall probability of correctness, we will consider all lengths of carry chains, starting from the most significant bit position. Let E_1 be the error probability caused by the shortest carry chain in the *most* significant $k+1$ bit positions, that is, starting at bit $n-k-1$ and propagating to bit $n-1$. Thus, this probability is (denoted as r as well)

$$E_1 = r = (\frac{1}{2})^k \cdot \frac{1}{4} = (\frac{1}{2})^{k+2} \tag{2}$$

Next term E_2 denotes the error probability caused by a carry chain originating at one bit to the right (bit position $n-k-2$), and the length of this carry chain can be either $k+1$ or $k+2$ to lead to an error. Each of the subsequent error probability terms will then cover additional "new" situations caused by the carry chain originating from the respective bit position, without overlapping with any of the previous situations already addressed. That is

$$E_i = r \cdot (1 - \sum_{j=1}^{i-k-1} E_j)$$

where r (see Eq. (2)) again represents the error probability that a carry chain of length of at least $k+1$ starts from this bit position and the summation term, $\sum_{j=1}^{i-k-1} E_j$, covers all the previous situations that overlap with this.

Further derivation leads to a closed-form expression for each E term:

$$E_i = r \cdot [\sum_{j}^{P} (-1)^{j-1} \cdot x_{i,j} \cdot r^j]$$

$$\text{where } r = (\frac{1}{2})^{k+2}, \quad P = \lceil \frac{i}{k+1} \rceil - 1 \quad \text{and} \quad x_{i,j} = \sum_{t=1}^{i-(k+1)j} \begin{pmatrix} j-2+t \\ j-1 \end{pmatrix}$$

And the overall error probability for an n-bit LAA using k-bit ACGBs, denoted as $E(n,k)$, is then a summation of all these mutually independent terms:

$$E(n,k) = \sum_{i}^{n-k} E_i \quad \text{since} \quad E_i = 0, \ \forall i > n-k$$

where there are not enough bits to have a $(k+1)$-bit carry chain.

Probability of an erroneous result from this approximation adder obviously depends on the value of k adopted relative to the size of operands n – the larger the k is the smaller the error rate $E(n,k)$ is, while the the rate increases with a larger n. The function $E(n,k)$ is plotted as in Fig. 1.

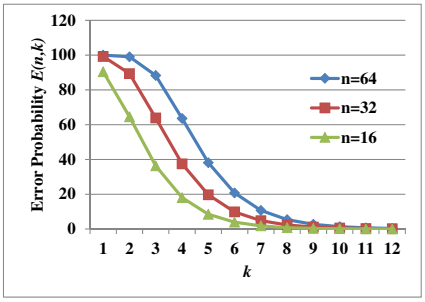


Fig. 1. Probability of erroneous result from an (n,k)-LAA.

Note that the benefits from this approximate adder are twofold: the overall cost is less than or comparable to all the fast parallel adders – total number of gates required is in the order of $O(n \times k)$ versus $O(n \log n)$ for others; the delay to reach the final result (albeit maybe incorrect) is constant (when k is kept constant) versus $O(\log n)$. A detailed comparison will be given in a later section.

All ACGBs produce their corresponding carry outputs in parallel after receiving the required p and g signals. Assume that circuit delay is measured in terms of delay of AND and OR gates (each presumed to be d) with inverter's delay ignored. If k is chosen small enough for a k-bit ACGB to generate its output with a simple two-level logic, that is, if k is no larger than the logic gates' fan-in limit, then the total delay for the LAA is $5d$: one d through the PGG circuits, $2d$ through all the ACGBs and $2d$ through the summation circuits. Even if k is selected to be larger than the fan-in limit, an extra $2d$ will be mostly sufficient to accommodate the necessary associativity gate expansion. Although this delay is much shorter than typical delays required for most fast parallel adders; for example, $25d$ is required for a 64-bit Brent-Kung adder (a complete comparison is to be given later),

most applications still demand the addition result to be correct at the end, which is not completely guaranteed by the proposed approximate adder. In order to accommodate this demand, the proposed design will incorporate an error detection and an error correction mechanisms into the adder to ensure a correct final result.

2.3 Error Detection

In the LAA design, an error occurs whenever there is a carry propagation chain longer than the block size k. The fastest way to detect such an error is to install a mechanism to detect a chain of $k+1$ bits, that is, the first (least significant) $k+1$ bits in the chain. This error detection signal can be derived from the two adjacent overlapping ACGB blocks which cover exactly these $k+1$ bits. For the sake of illustration, assume that these two adjacent blocks are the ones that produce ac_i and ac_{i-1}. Therefore, the $(k+1)$-bit chain is from

$$(g_{i-1}, p_{i-1}) = (g_{i-2}, p_{i-2}) = \ldots (g_{i-k-1}, p_{i-k-1}) = (0,1) \text{ AND } g_{i-k-2} = 1$$

which in turn leads to

$$ac_i = 0 \text{ and } ac_{i-1} = 1$$

in which $c_i \neq ac_i$ (which should have been a 1) is wrongly speculated. To quickly detect this, since both ac_i and ac_{i-1} are both available at the end of speculation process, we can use the following error detection logic to detect this error:

$$SE_i = \overline{ac_i} \cdot p_i \cdot ac_{i-1}.$$

This signal can detect an error caused by a $(k+1)$-bit carry propagation chain. If the error in ac_i is instead caused by a longer carry chain, then SE_i will not be able to detect it, which does not really pose a problem since there will be another detection signal in a less significant bit position along this chain that can detect the error which is caused by a chain of exactly $k+1$ bits. The overall error detection signal is then an OR combination of all these signals:

$$SE = SE_n + SE_{n-1} + \ldots + SE_{k+2} + SE_{k+1} \tag{3}$$

which requires a delay of $2d$ to obtain after all the approximate carries are available. Note that the delay of this error detection process may be hidden/absorbed if the final approximation addition result is correct since this process actually overlaps in time with the summation process which also takes $2d$. If the detection output eventually rises which indicates an error then another proposed add-on mechanism to be discussed next will proceed to correct the result.

2.4 Error Correction

All the carry bit errors can be easily corrected by adding one more product term in each ACGB to consider the carry input generated by the respective ACGB k bits to its right. Eq. 1 becomes

$$\begin{aligned} ac_i \quad = \quad & g_{i-1} + g_{i-2} \cdot p_{i-1} + g_{i-3} \cdot p_{i-1} \cdot p_{i-2} + \cdots \\ & + g_{i-k} \cdot p_{i-1} \cdot p_{i-2} \cdots p_{i-k+1} \\ & + p_{i-1} \cdot p_{i-2} \cdots p_{i-k+1} \cdot p_{i-k} \cdot ac_{i-k}. \end{aligned} \tag{4}$$

Note that the ACGBX that generates ac_i is provided with the carry ac_{i-k} as an additional input.

The pattern with which these ACGBXs are connected provides a total of k "parallel" error correction paths concurrently rectifying any wrong carry along the path, as shown in Fig. 2. For a carry propagation chain of length L ($L > k$), there are a total of $L-k$ approximate carry values incorrect from the original approximation process, and these $L-k$ carries will be corrected by the k correction paths in $\lceil \frac{L-k}{k} \rceil$ ACGBX "stages/cycles", with each such cycle incurring the delay of an ACGBX. Therefore, the maximum number of "stages/cycles" required to rectify all errors is, when $L = n$,

$$R = \lceil \frac{n-k}{k} \rceil = \lceil \frac{n}{k} \rceil - 1. \tag{5}$$

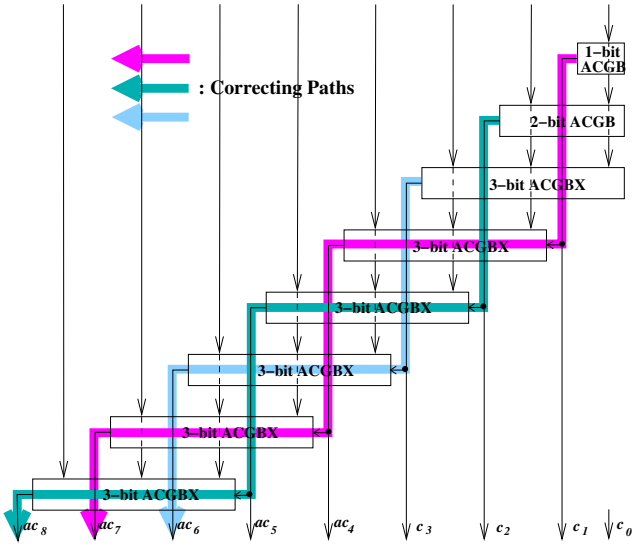


Fig. 2. k parallel error correcting paths with $k = 3$.

Note that this error correction process automatically starts whenever the parallel carry speculation process performed by all the ACGBXs is completed.

3. ASYNCHRONOUS PROCESSING

3.1 Completion Detection

The error detection signal SE (from Eq. (3)) will be used to indicate if there is an error from carry speculation. This signal will start with zero and be raised to one once at least an error occurs, and it will stay high until all the errors are corrected. This signal will be used to trigger the process completion indicator (C):

$$C = G \cdot \overline{SE}$$

in which the gating signal G will be used to "gate" the SE signal to prevent the initial low SE state to be wrongly interpreted as process completion. Starting from the time when operands are provided, the G signal will be forced to be low for at least $5d$ to prevent the low SE signal from passing through.

3.2 Delay Analysis

The total delay of the the proposed AAEC process, in the worst case, will require

$$\begin{aligned} D_{\text{worst}} &= D_{\text{PGG}} + D_{\text{ACGBX}} + D_{\text{Cor}} + D_{\text{Sum}} \\ &= d + 2d + R \cdot 2d + 2d \\ &= (2R+5)d \end{aligned}$$

where D_{PGG}, D_{ACGBX}, D_{Cor} and D_{Sum} each represents the delay for the PGG circuit, the ACGBX circuit, the correction process and the summation process, respectively. Note that R denotes the maximum number of correction stages as shown in Eq. (5). The range of the actual delay, denoted as D, is then

$$D_{\text{PGG}} + D_{\text{ACGBX}} + D_{\text{Sum}} \leq D \leq D_{\text{worst}} \quad \Longrightarrow \quad 5d \leq D \leq (2R+5)d. \tag{6}$$

The actual delay it takes to complete the process with a correct result depends on how long the correcting paths eventually take to rectify all errors. The longer the correcting path is required to correct the results, obviously the smaller its probability is. If this circuit is used in an asynchronous environment, its smaller "expected" (versus the worst-case) latency will further benefit the overall processing efficiency. The expected delay, denoted as D_{exp}, can be determined by:

$$D_{\text{exp}} = D_{\text{PGG}} + D_{\text{ACGBX}} + D_{\text{Cor\_exp}} + D_{\text{Sum}} \tag{7}$$

where $D_{\text{Cor\_exp}}$ denotes the expected delay for the correction process. The time chart for the whole process is illustrated in Fig. 3. In the best case, the AAEC produces the correct result without any error, which means the error detection signal SE never rises and thus the completion signal C will rise soon after $5d$. Since the summation process essentially overlaps with the error detection process in time, so the overall time required will be $5d$. On the other hand, if the error detection circuit produces a 1, then the signal will continue to be 1 until the error correction paths rectify all errors. Note that the correction process also is automatically initiated right after the approximate carries are generated. The expected delay from the correction process can be derived with:

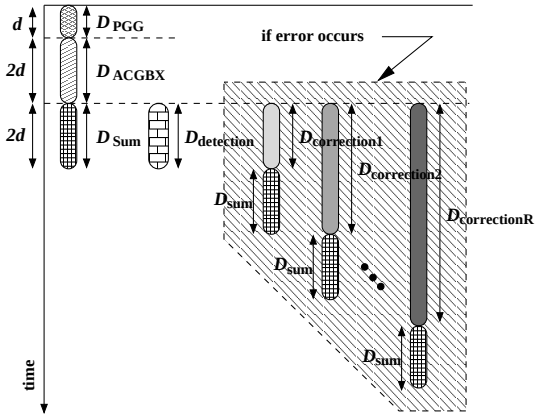


Fig. 3. Time graph for the approximate adder with correction process.

$$D_{\text{Cor\_exp}} = \sum_{s=0}^{R} D_{\text{Cor}_s} \cdot P_{E_s} \tag{8}$$

where D_{Cor_s} represents the correction delay required when the correction process goes through exactly s stages/cycles to rectify all errors, and P_{E_s} represents the probability of such occasion. Note that for the case of $s = 0$ there is no correction needed, whereas $s = R$ represents the situation where it requires the longest correction path. In general,

$$D_{\text{Cor}_s} = s \cdot 2d.$$

The corresponding P_{E_s} will be determined as follows. For the correction process to go through exactly s stages, the length of the longest carry chain, denoted as l, becomes

$$\lceil \frac{l-k}{k} \rceil = s$$

which leads to the following range for l:

$$s \cdot k + 1 \leq l \leq (s+1) \cdot k.$$

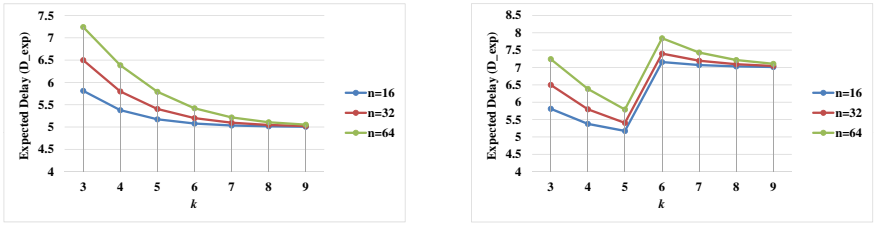
This then translates to the respective probability

$$P_{E_s} = E(n, s \cdot k) - E(n, (s+1) \cdot k).$$

For example, with $n = 32$ and $k = 5$, if the correction process is required to go through exactly 3 stages, then $16 \leq l \leq 20$, and its probability is exactly the difference between the error probability of having a $(32, 15)$-AAEC design and a $(32, 20)$-AAEC; that is,

$$P_{E_3} = E(32, 15) - E(32, 20).$$

Fig. 4 (a) shows the expected delay value (D_{exp}) of the proposed (n,k)-AAEC. Even with a tight fan-in limit of 5, the expected overall delay is still relatively small, under $8d$ throughout various k values. When k becomes larger the length of correction path becomes shorter thus leading to shorter delay. Fig. 4(b) shows the expected delay value of the proposed (n,k)-AAEC under a fan-in of 5. Such a fan-in limit leads to the jump from $k = 5$ to $k = 6$ since any gate requiring more than 5 inputs will need one more level of OR gate.



(a) (b)

Fig. 4. Expected delay value of the (n,k)-AAEC with (a) No fan-in limit and (b) with a fan-in limit of 5.

3.3 Cost Analysis

Throughout the rest of this paper, for the sake of simplicity, the cost of a design is in terms of the number of logic gates (AND and OR) required. The number of logic gates required for an (n,k)-AAEC, denoted as $N_{\text{AAEC}}(n,k)$, can be roughly derived by

$$\begin{aligned} N_{\text{AAEC}}(n,k) &= N_{\text{PGG}} + N_{\text{ACGBX}} + N_{\text{Sum}} + N_{\text{Detection}} \\ &\approx 2n + (k+1) \cdot n + 4n + (n-k+1) \\ &= nk + 8n - k + 1 \end{aligned}$$

4. COMPARISON

In this section the delay and cost required for the proposed AAEC design is compared to some of the most widely recognized and adopted parallel adders, the Kogge-Stone adder [1] and the Brent-Kung adder [2], and one speculative design in [13].

4.1 Delay Comparison

The Kogge-Stone adder can generate all carry signals in $O(\log n)$ time, and is widely considered one of the fastest adder designs. Its delay, denoted as $D_{\text{K-S}}(n)$, represented in terms of d, is

$$\begin{aligned} D_{\text{K-S}}(n) &= D_{\text{PGG}} + D_{\text{Carry-Tree}} + D_{\text{Sum}} \\ &= d + (\log_2 n) \cdot 2d + 2d \\ &= (2\log_2 n + 3)d \end{aligned}$$

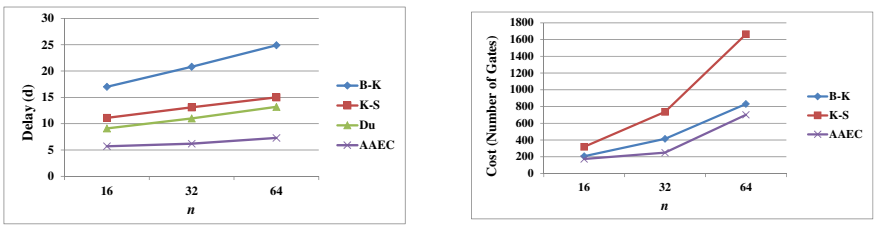
where the $\log_2 n$ is the number of levels of the carry-generating tree.

The Brent-Kung adder design is similar to the Kogge-Stone one by trading off delay for the benefit of lower cost. Its delay, denoted as $D_{\text{B-K}}$, represented in terms of d, is

$$\begin{aligned} D_{\text{B-K}}(n) &= D_{\text{PGG}} + D_{\text{Carry-Tree}} + D_{\text{Sum}} \\ &= d + (2\log_2 n - 1) \cdot 2d + 2d \\ &= (4\log_2 n + 1)d. \end{aligned}$$

The speculative adder presented in [13] (to be referred to as the Du adder), require a much longer speculative processing time ($\approx 4d \log k$) than the proposed AAEC (a very small constant delay ($\approx 3d$) and a substantially more complex and costly correction logic ($\approx 3\frac{n}{k}\log\frac{n}{k}$ versus k in ours).

Fig. 5 (a) displays the delay comparison of the proposed AAEC with $k = 3$ against the three other adder designs. The logarithmic slope of the AAEC's delay is estimated to be 0.7 which is much smaller than the others, 2 for Kogge-Stone and 4 for Brent-Kung. Even with a very modest k value ($k = 3$), the improvement in delay is already very significant. A larger k will further lower the slope of AAEC's delay – down to about 0.2 for $k = 6$, albeit with the concern of physical limitation.



(a) (b)

Fig. 5. Delay comparison (a) and cost comparison (b).

4.2 Cost Comparison

Number of gates required for the n-bit Kogge-Stone adder, denoted as $N_{\text{K-S}}(n)$ can be roughly estimated as

$$N_{\text{K-S}}(n) \approx 5n + [(\log_2 n) \cdot 3n - n] + 4n = 3n\log_2 n + 8n.$$

Number of gates required for the n-bit Brent-Kung adder, denoted as $N_{\text{B-K}}(n)$ can be roughly estimated as

$$N_{\text{B-K}}(n) \approx 5n + (4n-1) + 4n = 13n - 1.$$

Fig. 5 (b) displays the delay comparison of the proposed AAEC with $k = 3$ against the two parallel adders. Note that the adder proposed in [13] adopts a sequential process for error recovery without clearly specifying the logic required, thus it is not included in the cost comparison. Again, even with a very modest k value ($k = 3$), the cost of AAEC roughly reflects a 16% saving compared to that of Brent-Kung and over 50% versus Kogge-Stone in these cases. An AAEC with a larger k will further increase the improvement percentages.

5. CONCLUSION

A fast and cost effective approximate adder design was proposed in this paper. Not only delay is shortened but the overall cost is reduced with this design compared to the most prevalent fast adders. Power consumption is also shown to be lower under various VLSI technologies. The potential of the proposed design is further enhanced by its capability to function in an asynchronous environment. This research can obviously be further extended by trading off correction latency for lower cost by adjusting the correction path connections, which should become a very useful design parameter in order to satisfy various design requirement specifications.

ACKNOWLEDGEMENT

This research is partially sponsored by National Science Foundation Award No. CNS-1538418.

REFERENCES

1. P. Kogge and H. Stone, "A parallel algorithem for the efficient solution of a general class of recurrence relation," *IEEE Transactions on Computers*, Vol. C-22, 1973, pp. 786-793.
2. R. Brent and H. Kung, "A regular layout for parallel adders," *IEEE Transactions on Computers*, Vol. C-31, 1982, pp. 260-264.
3. J. Huang, J. Lach, and G. Robins. "A methodology for energy-quality trade-off using imprecise hardware," in *Proceedings of the 49th Design Automation Conference*, 2012, pp. 504-509.
4. Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi, "Approximate XOR/XNOR-based adders for inexact computing," in *Proceedings of IEEE International Conference on Nanotechnology*, 2013, pp. 690-693.
5. J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proceedings of the 18th IEEE European Test Symposium*, 2013, pp. 1-6.
6. V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Transactions on Computer-Aided Design Integrated Circuits and Systems*, Vol. 32, 2013, pp. 124-137.

7. T. Liu and S. Lu, "Performance improvement with circuit-level speculation," in *Proceedings of International Symposium on Microarchitetcure*, 2000, pp. 348-355.
8. S.-L. Lu, "Speeding up processing with approximation circuits," *Computer*, Vol. 37, 2004, pp. 67-73.
9. A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: A new paradigm for arithmetic circuit design," in *Proceedings of Design Automation and Test in Europe*, 2008, pp. 1250-1255.
10. A. Cilardo, "A new speculative addition architecture suitable for two's complement operations," in *Proceedings of Design Automation and Test in Europe*, 2009, pp. 664-669.
11. Baneres *et al*., "Variable-latency design by function speculation," in *Proceedings of Design, Automation and Test in Europe*, 2009, pp. 1704-1709.
12. Y. Liu *et al*., "Design methodology of variable latency adders with multistage function speculation," in *Proceedings of International Symposium on Quality Electronic Design*, 2010, pp. 824-830.
13. K. Du, P. Varman, and K. Mohanram, "High performance reliable variable latency carry select addition," in *Proceedings of Design Automation and Test in Europe*, 2012, pp. 1257-1262.
14. M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *Proceedings of the 52nd ACM/EDAC/IEEE Design Automation Conference*, 2015, pp. 1-6.
15. A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Proceedings of Design Automation Conference*, 2012, pp. 820 -825.
16. R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *Proceedings of International Conference on Computer-Aided Design*, 2013, pp. 48-54.
17. V. Camus, J. Schlachter, and C. Enz, "Energy-efficient inexact speculative adder with high performance and accuracy control," in *Proceedings of IEEE International Symposium on Circuits and Systems*, 2015, pp. 48-54.
18. M. Weber, M. Putic, H. Zhang, J. Lach, and J. Huang, "Balancing adder for error tolerant applications," in *Proceedings of IEEE International Symposium on Circuits and Systems*, 2013, pp. 3038-3041.
19. D. Esposito, D. De Caro, E. Napolu, N. Petra, and A. G. M. Strollo, "Variable latency speculative Han-Carlson adder," *IEEE Transactions on Circuits and Systems*, Vol. 62, 2015, pp. 1353-1361.
20. S. Mazahir, O. Hasan, R. Hafiz, M. Shafique, and J. Henkel, "Probabilistic error modeling for approximate adders," *IEEE Transactions on Computers*, Vol. 66, 2017, pp. 515-530.
21. I. C. Lin, Y. M. Yang, and C. C. Lin, "High-performance low-power carry speculative addition with variable latency," *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 23, 2015, pp. 1591-1603.

Tarak K. Kodali earned his bachelor's degree in Electronics and Communication Engineering from Vasireddy Venkatadri Institute of Technology, Inida in 2014. He then received his M.S. degree in Electrical Engineering from the University of Texas at San Antonio (UTSA) in 2016. He is currently a software developer with Deloitte Consulting at New York.

Yilin Zhang received her B.S. and M.S. degrees in Electrical Engineering from Dalian Maritime University in 2007 and Beijing University of Posts and Telecommunications, Beijing in 2010, respectively. She then received her Ph.D. degree in Electrical Engineering from the University of Texas at San Antonio (UTSA) in 2014. She has been with Advanced Micro Devices as a Member of Technical Staff since then. Her research interests include computer architecture and parallel and distributed computing. She has published over a dozen papers in international conference proceedings and journals.

Eugene John received his Ph.D. in Electrical Engineering from the Pennsylvania State University. He is currently a Professor in the Department of Electrical and Computer Engineering at the University of Texas at San Antonio. His research interests include energy efficient computing, energy efficient hardware for deep learning, hardware security, low power VLSI systems, power aware cloud computing, computer architecture and benchmarking. He is a recipient of the University of Texas System Regent's Outstanding Teaching Award in 2014.

Wei-Ming Lin is currently a Professor of Electrical and Computer Engineering at the University of Texas at San Antonio (UTSA). Dr. Lin received the BS degree in Electrical Engineering from National Taiwan University, Taipei, Taiwan, in 1982, the M.S. and Ph.D. degrees in Electrical Engineering from the University of Southern California, Los Angeles, in 1986 and 1991, respectively. Dr. Lin's research interests are in the areas of distributed and parallel computing, computer architecture, computer networks, autonomous control and internet security. He has published over 120 technical papers in various conference proceedings and journals. Dr. Lin's past and on-going research has been supported by NSF, DOD, ONR, AFOSR, *etc*.