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

Tarak K. Kodali\textsuperscript{1}, Yilin Zhang\textsuperscript{2}, Eugene John\textsuperscript{1} and Wei-Ming Lin\textsuperscript{1}
\textsuperscript{1}Department of Electrical and Computer Engineering
University of Texas at San Antonio
San Antonio, TX 78249
\textsuperscript{2}Advanced Micro Devices, Inc.
Austin, TX 78735, USA

July 25, 2018

Abstract

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. Dynamic power consumption is also shown to favor the proposed design over others. The proposed design is further implemented using H-Spice and tested to verify our analysis and comparison results.

Keywords: Approximate Adder; Speculative Adder; Speculative Circuit; Asynchronous Adder; Variable-Latency Adder

1 Introduction

Circuit miniaturization has been continuously providing the means for enhancing performance of microprocessors, accompanied by the evolution of processing paradigm, from instruction pipelining, superscalar, multi-threading, speculative processing, among many others. Most of the fundamental operations performed by a microprocessor are dependent on integer additions, including all other integer arithmetic operations and all floating point operations. Processing efficiency of other program controlling and speculative instructions, such as jumps and branches, also depends on the integer
addition operation. Essentially clock speed of a processor or an ASIC chip very much depends on how fast an addition can be performed.

There have been many adder designs proposed in the literature and employed in various systems. Simpler and serial designs, such as Ripple Carry Adder, Carry Skip Adder, among many others are not at all popular due to their slow processing speed, which is limited by the potentially long chain of carry dependency. Parallel (or semi-parallel) adders have been dominating all various applications where 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, 4, 5, 6, 7, 8].

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 [9, 10, 11]. Additional circuitries have been also established to provide error detection and correction in [12, 13, 14, 15, 16, 17, 18]. Configurable-accuracy adders to reduce the delay of the traditional adder are proposed in [19, 20] 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 [21] which further attempted to improve hardware utilization efficiency, or [22] in which error resilience is improved, and [23] in which error probability is reduced using a hierarchical design approach. A detailed probabilistic model for analyzing error occurrence is given in [24]. A design by separating carry generator and sum generator to reduce power consumption is proposed in [25]. Another low-power design is presented in [26] with a statistical analysis to obtain it error rate. A survey of several approximate adders and their trade-off between accuracy and area are given in [27]. A comprehensive study on approximate computing in general methodologies and optimization framework is given in [28].

In one of the most advanced approximate adder designs [17], addition result is speculated using windows of non-overlapping bits to reduce overall logic requirement. The design is based on a “modular” Kogge-stone adder (or a similar fast parallel prefix adder) with each of non-overlapping modular blocks using the speculative carry generate output of its previous block to speculate its own results. Once an error is detected, another level of parallel adder processing tree is applied to correct the results. Compared to the one proposed in [10] which uses overlapping windows (blocks), such a “two-step” speculative process requires a much longer time to produce its speculated results, though with a higher accuracy probability. The error correction logic included is also not insignificant, requiring another smaller-size parallel adder. In addition, the error rate analysis performed in [17] is wrongly conducted by assuming mutual exclusiveness among events which are actually mutually
dependent, which leads to an incorrect assessment of the overall performance effectiveness of the design. Furthermore, due to its non-overlapping nature, it also proposes a more complex design for signed addition to address the different bit behavior from two’s complement numbers, a non-existing drawback in designs with overlapping blocks. With all these issues, designing a faster and simpler approximate adder is eminent to the further advancement of the overall approximate computing research and applications.

This paper (based on [29]) proposes an approximate adder design based on a simple bit-overlapping design in [10], 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

In this section the proposed Approximate Adder with Error Correction (AAEC) is described. For the sake of completeness, the original approximate adder design which our design in based on is also briefly presented.

2.1 The Lu Approximate Adder

The proposed approximate adder design is based on a well-known approximate adder, the Lu Approximate Adder (LAA) [10]. 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:

\[
ac_i = g_{i-1} + g_{i-2} \cdot p_{i-1} + g_{i-3} \cdot p_{i-1} \cdot p_{i-2} + \cdots + g_{i-k} \cdot p_{i-1} \cdot p_{i-2} \cdots p_{i-k+1}
\]

(1)

or

\[
ac_i = \sum_{u=1}^{k} (g_{i-u} \cdot \prod_{v=1}^{u-1} p_{i-v})
\]

where the summation and product notations are borrowed for the corresponding logical OR and AND operations, respectively. A block diagram of a \( k \)-bit ACGB is shown in Figure 1, in which PGG denotes the logic to generate the \( p \) and \( g \) signals for each bit position. 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. An 8-bit LAA design using 3-bit ACGBs is shown in Figure 2.

### 2.2 Error Probability Analysis

The error probability analysis in [10] 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 Figure 2 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. An error occurs when such a chain occurs as shown in Figure 3.

Figure 3: An Error Occurs with a Carry Propagation Chain Longer than $k$

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 in Figure 2, $ac_5$ is incorrect with the probability of

$$\left(\frac{1}{2}\right)^k \cdot \frac{1}{4} + \left(\frac{1}{2}\right)^k \cdot \frac{1}{2} \cdot \frac{1}{4}$$

where $k = 3$, 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)

\[
E_1 = r = \left( \frac{1}{2} \right)^k \cdot \frac{1}{4} = \left( \frac{1}{2} \right)^{k+2}
\]  

(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. The relationship among these cases is demonstrated in Figure 4 with \( n = 32 \) and \( k = 3 \). Obviously \( E_2 \) is equal to \( r \) as well, and situations for \( E_2 \) is independent of those for \( E_1 \). This is clearly illustrated by the example in Figure 4 where cases for \( E_1 \) require that input bits \((a_{28}, b_{28})\) to be \((1, 1)\) for its carry chain initiation while those for \( E_2 \) require these two input bits to be either \((1, 0)\) or \((0, 1)\) for carry propagation. 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

\[
E_i = r \cdot \left( 1 - \sum_{j=1}^{i-k-1} E_j \right)
\]

where \( r \) (see Equation 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. For the example in Figure 4, analysis of \( E_9 \) leads to a result of

\[
E_9 = r \left[ 1 - (E_1 + E_2 + E_3 + E_4 + E_5) \right]
\]

for \( k = 3 \). Note that \( E_9 \) represents the error probability from the respective additional new cases caused by a carry chain starting at bit position 20. These new cases correspond to the error probability of \( r \) (from its own carry chain of at least \((k + 1)\) bits), excluding all the cases that have been covered which are from the cases spanning from \( E_1 \) to \( E_5 \). Cases for \( E_6, E_7 \) and \( E_8 \) are independent of those for \( E_9 \); that is, they are all mutually exclusive since they differ with one another by at least by one bit position in their own respective carry-chain’s starting bit position; namely, \((a_{23}, b_{23}) = (1, 1)\) for \( E_6 \),
(a_{22}, b_{22}) = (1, 1) for E_7 and (a_{21}, b_{21}) = (1, 1) for E_8, while E_9 requires these bits to be otherwise instead for carry propagation.

The discussion above leads to the following list of expressions for some E values for the case of \( k = 3 \):

\[
\begin{align*}
E_1 &= r \\
E_2 &= r \\
E_3 &= r \\
E_4 &= r \\
E_5 &= r(1 - r) \\
E_6 &= r(1 - 2r) \\
E_7 &= r(1 - 3r) \\
E_8 &= r(1 - 4r) \\
E_9 &= r(1 - 5r + r^2) \\
E_{10} &= r(1 - 6r + 3r^2) \\
E_{11} &= r(1 - 7r + 6r^2) \\
E_{12} &= r(1 - 8r + 10r^2) \\
E_{13} &= r(1 - 9r + 15r^2 - r^3) \\
E_{14} &= r(1 - 10r + 21r^2 - 4r^3) \\
E_{15} &= r(1 - 11r + 28r^2 - 10r^3) \\
E_{16} &= r(1 - 12r + 36r^2 - 20r^3) \\
E_{17} &= r(1 - 13r + 45r^2 - 35r^3 + r^4) \\
E_{18} &= r(1 - 14r + 55r^2 - 56r^3 + 5r^4) \\
E_{19} &= r(1 - 15r + 66r^2 - 84r^3 + 15r^4) \\
E_{20} &= r(1 - 16r + 78r^2 - 120r^3 + 35r^4)
\end{align*}
\]

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

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

where \( r = (\frac{1}{2})^{k+2} \), \( P = \lceil \frac{i}{k+1} \rceil - 1 \) and

\[
x_{i,j} = \sum_{t=1}^{i-(k+1)j} \binom{j - 2 + t}{j - 1}
\]

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:

\[ E(n, k) = \sum_{i}^{n-k} E_i \]

since

\[ E_i = 0, \quad \forall i > n - k \]

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 Figure 5. For

\[ n = 16, \text{ the probability for the circuit to produce a correct result exceeds 91\% when } k \text{ is set to be 5 or higher, while it takes } k \geq 7 \text{ to achieve the same feat for } n = 64. \text{ Note that this theoretical result is correctly verified by our simulation using 100,000 randomly generated input values, with a negligible discrepancy. The original analysis in [10] instead produced a result underestimating the error probability by up to 3\% (see Figure 6).} \]

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
Lu’s analysis
Correct analysis

Figure 6: Comparison of the Incorrect Results from [10] with Our Analysis on Error Probability for an \((n, k)\)-LAA with \(n = 64\)

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

\[
(g_{i-1}, p_{i-1}) = (g_{i-2}, p_{i-2}) = \ldots (g_{i-k-1}, p_{i-k-1}) = (0, 1),
\]

AND \(g_{i-k-2} = 1\)

which in turn leads to

\[\quad ac_i = 0 \text{ and } ac_{i-1} = 1\]

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:

\[
SE_i = \overline{ac_i} \cdot p_i \cdot ac_{i-1}
\]

Figure 7 shows an example of two adjacent 5-bit ACGBs leading to \(ac_9 = 0\) and \(ac_8 = 1\), an error to be detected with \(SE_9 = \overline{ac_9} \cdot p_8 \cdot ac_8 = 1\). 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
Figure 7: Error Detection Using Two Adjacent ACGBs

Figure 8: Error Detection Circuitry for the Approximate Adder with \( n = 8 \) \( k = 3 \)

The overall error detection signal is then an OR combination of all these signals:

\[
SE = SE_n + SE_{n-1} + \ldots + SE_{k+2} + SE_{k+1}
\]  

(3)

which requires a delay of \( 2d \) to obtain after all the approximate carries are available. The overall error detection circuitry is shown in Figure 8 for a case of \( n = 8 \) and \( k = 3 \). 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. Equation 1 becomes

\[
ac_i = g_{i-1} + g_{i-2} \cdot p_{i-1} + g_{i-3} \cdot p_{i-1} \cdot p_{i-2} + \cdots \\
+ g_{i-k} \cdot p_{i-1} \cdot p_{i-2} \cdots p_{i-k+1} \\
+ p_{i-1} \cdot p_{i-2} \cdots p_{i-k+1} \cdot p_{i-k} \cdot ac_{i-k}
\]  

(4)
This slightly modified ACGB is denoted as ACGBX and is shown in Figure 9. Note that the ACGBX that generates \( ac_i \) is provided with the carry \( ac_{i-k} \) as an additional input. The complete design with these modifications is shown in Figure 10 and is denoted as Approximate Adder with Error Correction (AAEC), and \((n, k)\)-AAEC is used to denote an \( n \)-bit AAEC with \( k \)-bit ACGBXs.

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 Figure 11. 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 \( L = n \),

\[
R = \left\lceil \frac{n-k}{k} \right\rceil = \left\lceil \frac{n}{k} \right\rceil - 1
\]
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 Equation 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. A simplified time chart example is given in Figure 12.

3.2  Delay Analysis

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

\[
D_{\text{worst}} = D_{\text{PGG}} + D_{\text{ACGBX}} + D_{\text{Cor}} + D_{\text{Sum}}
\]

\[
= d + 2d + R \cdot 2d + 2d
\]

\[
= (2R + 5)d
\]
where $D_{\text{PGG}}, D_{\text{ACGBX}}, D_{\text{Cor}}$ and $D_{\text{Sum}}$ 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 Equation 5. The range of the actual delay, denoted as $D$, is then

$$D_{\text{PGG}} + D_{\text{ACGBX}} + D_{\text{Sum}} \leq D \leq D_{\text{worst}}$$

$$5d \leq D \leq (2R + 5)d$$

(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 $D_{\text{exp}}$, can be determined by:

$$D_{\text{exp}} = D_{\text{PGG}} + D_{\text{ACGBX}} + D_{\text{Cor,exp}} + D_{\text{Sum}}$$

(7)

where $D_{\text{Cor,exp}}$ denotes the expected delay for the correction process. The time chart for the whole process is illustrated in Figure 13. In the best case, the AAEC produces the correct result without any error, which means the error detection signal $SE$ (see Figure 8) never rises and thus the comple-
tion 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:

$$D_{\text{Cor,exp}} = \sum_{s=0}^{R} D_{\text{Cor}_s} \cdot P_{E_s}$$

where $D_{\text{Cor}_s}$ 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

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

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,

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

One complete example is shown in Table 1 for $n = 32$ and $k = 5$. The expected correction delay for this example using Equation 8 is then derived to be $0.4053d$ which leads to a total delay of $5.4053d$. Figure 14 shows the probability $P_{E_s}$, versus $k$ under different $n$ values.

Figure 15 shows the expected delay value ($D_{\text{exp}}$) 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. Figure 16 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.
Table 1: The Process to Determine the Expected Delay of a (32, 5)-AAEC

<table>
<thead>
<tr>
<th>s</th>
<th>$D_{Cor}$</th>
<th>$P_{E_s}$(%)</th>
<th>$D_{Cor} \cdot P_{E_s}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$E(32,0) - E(32,5) = 80.28392$</td>
<td>$0d$</td>
</tr>
<tr>
<td>1</td>
<td>2d</td>
<td>$E(32,5) - E(32,10) = 19.17936$</td>
<td>$0.3836d$</td>
</tr>
<tr>
<td>2</td>
<td>4d</td>
<td>$E(32,10) - E(32,15) = 0.52375$</td>
<td>$0.0209d$</td>
</tr>
<tr>
<td>3</td>
<td>6d</td>
<td>$E(32,15) - E(32,20) = 0.01268$</td>
<td>$0.0007d$</td>
</tr>
<tr>
<td>4</td>
<td>8d</td>
<td>$E(32,20) - E(32,25) = 0.00028$</td>
<td>$\approx 0d$</td>
</tr>
<tr>
<td>5</td>
<td>10d</td>
<td>$E(32,25) - E(32,30) = 0.00001$</td>
<td>$\approx 0d$</td>
</tr>
<tr>
<td>6</td>
<td>12d</td>
<td>$E(32,30) - E(32,32) = 0.00000$</td>
<td>$\approx 0d$</td>
</tr>
</tbody>
</table>

$D_{Cor,exp} = 0.4053d$
Figure 14: Probability of Requiring Different Correction Lengths ($P_{E_s}$) for (a) $n = 16$ (b) $n = 32$ (c) $n = 64$
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

\[
N_{\text{AAEC}}(n, k) = N_{\text{PGG}} + N_{\text{ACGBX}} + N_{\text{Sum}} + N_{\text{Detection}} \\
\approx 2n + (k + 1) \cdot n + 4n + (n - k + 1) \\
= nk + 8n - k + 1
\]

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 [17].
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 $D_{K-S}(n)$, represented in terms of $d$, is

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

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 $D_{B-K}$, represented in terms of $d$, is

$$D_{B-K}(n) = D_{PGG} + D_{Carry-Tree} + D_{Sum}$$
$$= d + (2 \log_2 n - 1) \cdot 2d + 2d$$
$$= (4 \log_2 n + 1)d$$

The speculative adder presented in [17] (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 ($\approx 3n \log \frac{n}{k}$ versus $k$ in ours).

Figure 17 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.

4.2 Cost Comparison

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

$$N_{K,S}(n) \approx 5n + [(\log_2 n) \cdot 3n - n] + 4n$$
$$= 3n \log_2 n + 8n$$
Number of gates required for the $n$-bit Brent-Kung adder, denoted as $N_{B-K}(n)$ can be roughly estimated as

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

Figure 18 displays the delay comparison of the proposed AAEC with $k = 3$ against the two parallel adders. Note that the adder proposed in [17] 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 Implementation

To further verify our analysis and comparison results, the proposed AAEC as well as the Kogge-Stone and Brent-Kung adders are all implemented using H-Spice and tested with various input test vectors, and simulated in 45nm, 65nm, 90nm and 180nm technologies. The size of all adders tested is set to $n = 32$, and the ACGBs used in the tested AAEC are of size $k = 5$.

5.1 Timing and Functional Verification of AAEC

To verify the time graph displayed in Figure 12 for process completion of the AAEC circuit, an H-Spice simulation is first run in a 45nm technology on our $(32, 5)$-AAEC design. From Equation 6, such a design will incur a delay $D$ with the range of

$$5d \leq D \leq [(2\lceil \frac{32}{5} \rceil - 1) + 5]d = 18d$$

A test input which requires a correction path of 3 stages is first given to the simulator. Note that the gate delay $d$ under this environment is roughly 30ps. As shown in Figure 19, the signal at the bottom is the start signal used to trigger the addition process, and the one next to the bottom is the gating signal $G$ being delayed by roughly $6d$ (instead of $5d$ to guard against gate delay variations). We can
see that the wave form of error signal $SE$ (the second from the top) rises to 1 (due to the given test input with speculative errors) right after about $5d$. Once $SE$ falls down which indicates all errors have been corrected and the completion signal $C$ (the top one) rises immediately after. Note that the duration of $SE = 1$ is roughly about $6d$ which confirms our delay estimation for the correction process, a correction path through 3 ACGBXs on the given test input.

Another example in Figure 20 displays the case where there is no error from the speculative process, in which the $SE$ signal stays low and the completion signal rises right after $6d$ when the gating signal $G$ is up.

### 5.2 Power Consumption Comparison

Another key aspect of our comparison is in the power consumption, including both the dynamic power and leakage power dissipation. A $(32, 5)$-AAEC is compared to both the Kogge-Stone and Brent-Kung adders of 32 bits. Power consumption of an AAEC depends on how long it takes to rectify its errors. Although, for a $(32, 5)$-AAEC, more than 80% of time it does not require any correction, we choose an input test case that requires 3 stages of correction process for comparison just to illustrate the superiority of the AAEC even when the probability of a correction that takes at least 3 stages is less than 0.013% (see Table 1). Comparison results on dynamic power consumption among the three designs are shown in Figure 21. Even with a case that drastically overestimates the power consumption for the AAEC, it still displays a saving of about 30 to 40% compared to the other two. For cases that are more likely to happen which require less (or even no) correction time, the power saving will be even much more significant. Comparison on the leakage power consumption is shown in Figure 22. All three are comparable to one another in this aspect, which depends on how often the circuit is utilized and if there is any built-in mechanism to reduce this power consumption.
Figure 20: AAEC Error Detection and Completion Signals when Error Correction Is Not Required Using H-Spice with 45nm Technology. Wave Forms Displayed (from the bottom): Start Signal ($S$), Gating Signal ($G$), Error Signal ($SE$), and Completion Signal($C$)

Figure 21: Dynamic Power Consumption Comparison on A Special Test Vector

Figure 22: Leakage Power Consumption Comparison on A Special Test Vector
Another set of simulations are run on ten random test vectors, with the average power dissipation shown in Figure 23 and Figure 24. As expected, the improvement in dynamic power with the proposed AAEC over the other fast adders is more significant on the average due to the variable latency capability of the former.

5.3 Delay Comparison

To verify the analytical delay comparison result from Figure 17, actual delay data are obtained using the same H-Spice environment with the same parameters: a (32, 5)-AAEC compared with 32-bit Kogge-Stone and Brent-Kung adders. Again an input that requires 3 stages of correction path for the AAEC is used. The results are shown in Figure 25. Again, even with a very unlikely long delay-requiring input case, the AAEC still runs faster than the other two, 17% faster than Kogge-Stone adder and 37% faster than Brent-Kung adder, which ascertains the analytical results.

Average delay values from the ten random test vectors are shown shown in Figure 23. Note that the average delay of the proposed AAEC adder has significantly decreased compared to the special test case, showing a much more significant improvement over the other two adders.
6 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. Limitations of this research are in that the proposed AAWC adder does not out-perform the Brent-Kung adder by much in cost, and that the design does not compare favorably in execution time to a pure approximate adder without any concern for correction. 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.

7 Acknowledgement

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


