Sensor-Based Approximate Adder Design for Accelerating Error-Tolerant and Deep-Learning Applications

Ning-Chi Huang, Szu-Ying Chen and Kai-Chiang Wu
Department of Computer Science
National Chiao Tung University, Hsinchu, Taiwan
nchuang@cs.nctu.edu.tw, sandy81917@gmail.com, kcw@cs.nctu.edu.tw

***Abstract*—Approximate computing is an emerging strategy which trades computational accuracy for computational cost in terms of performance, energy, and/or area. In this paper, we propose a novel sensor-based approximate adder for high-performance energy-efficient arithmetic computation, while considering the accuracy requirement of error-tolerant applications. This is the first work using in-situ sensors for approximate adder design, based on monitoring online transition activity on the carry chain and speculating on carry propagation/truncation. On top of a fully-optimized ripple-carry adder, the performance of our adder is enhanced by 2.17X. When applied in error-tolerant applications such as image processing and handwritten digit recognition, our approximate adder leads to very promising quality of results compared to the case when an accurate adder is used.**

I. Introduction

In the recent decades, with the relentless miniaturization of semiconductor process technologies, improving performance or energy efficiency of integrated circuits has become more and more challenging. On the other hand, computing systems are embedded in various kinds of machines, and mobile devices are widespread and equipped with more advanced features which are hungry for high-speed, low-power computing resources. As data volume grows exponentially, software applications become more data-driven, further driving the need of data analysis/prediction for finding helpful information or hidden rules among seas of data. Toward this end, artificial intelligence, machine learning, and deep learning have made huge progress in a variety of domains, from technology to daily life; however, they involve intensive data computation and may consume excessive energy and time. For example, AlphaGo [1] defeated the world champion in Go match with 1920 CPUs and 280 GPUs running on its distributed platform.

Most of the existing data-driven applications have the following characteristics: (*i*) The golden solution is difficult to be defined, e.g., web search does not have an exact solution given a certain search query and many less-than-perfect results are acceptable. Also, due to the imperfect perception of human beings, an image can be compressed and not noticed by people. (*ii*) For some applications, the perfect answer is not always possible to find because of imperfect input data, e.g., audio data which is recorded along with environmental noise and distorted again after applying analog-to-digital conversion. (*iii*) Obtaining the perfect answer is too expensive. Therefore, Monte-Carlo-based methods, such as Monte-Carlo tree search exploited by AlphaGo, which rely on repeated random sampling to obtain results, are very useful when other computationally-expensive approaches are not applicable.

It can be noted that most data-driven applications are characteristic of inherent error tolerance; that is, they accept high-quality or nearly correct (if hardly perfect) results. As long as the quality of results can be maintained, these error-tolerant applications do not require extreme effort of computation, and trade-offs can be made between computational accuracy (in terms of quality of results) and computational cost (in terms of energy and time). Therefore, different levels of *approximate computing* strategies are presented. With marginal loss of computational accuracy, they can speed up the circuit or reduce circuit area and power consumption. In general, there are three types of approximation techniques for improving performance and energy efficiency [2]. The first technique is voltage over-scaling; although it may violate timing constraints and induce errors, it is the most effective way to reduce power consumption [3][4]. The second technique is generating approximate circuits based on quality constraints by logic synthesis [5]-[8].

The other technique is redesigning (or simplifying) the circuit into an approximate version according to its usage or specification. For machine learning applications, which are usually based on numerous matrix manipulations, simplifying a significant fraction of "exact" arithmetic operations into "approximate" ones is an elegant method of trading accuracy for energy and computation time. Recently, various approximate adders have been proposed. IMPACT [9] redesigns the transistor-level structure of an adder into an approximate version to provide energy saving. ETAI [10] splits the adder into an accurate part and an inaccurate part, where normal addition and reverse simplified addition are performed respectively, bi-directionally. Accuracy Configurable Adder (ACA) [11], ETAII [12] and Almost Correct Adder (ACA*) [13] overlap multiple sub-adders to form a sum-generation portion and a carry-generation portion in aid of accuracy requirement. Speculative Carry Select Adder (SCSA) [14] segments the chain of carry propagation signals into blocks of the same size and speculates the carry bits between blocks. CCB [15] uses Carry Cut-Back technique to prevent the activation of long carry propagation, thereby relaxing the timing constraints of the design. To reduce area overhead, QuAd [16] consists of multiple non-overlapping sub-adders, in contrast to [11]-[13] where multiple (and larger number of) sub-adders are overlapped. From these previous studies, we can notice that most of them take advantage of the fact that the actual carry propagation is frequently much shorter than the length of the entire adder. Hence, arithmetic addition can be accelerated and acceptable results can be produced by employing multiple sub-adders with shorter bit-width.

However, those previous approximate adders with multiple overlapping sub-adders have large area overhead, the ones with smaller area overhead cannot ensure high quality of results due

* The authors of both [11] and [13] presented and abbreviated their approximate adders as ACA.

978-3-9819263-2-3/DATE19/©2019 EDAA

to sporadic but excessive loss of computational accuracy, and the ones with satisfactory accuracy loss hardly achieve high performance. In this paper, we propose an approximate adder architecture in which performance, computational accuracy and area overhead are all taken into account.

The contributions and advantages of our work are as follows:

- ***Significant performance improvement with limited area overhead and high, configurable computational accuracy*:** This is the first work simultaneously exploiting *both approximate computing for accelerating error-tolerant applications and variable-latency design for further performance enhancement*. For the first time, based on using an in-situ sensor to monitor online transition activity on the carry chain and speculate on carry propagation/truncation, we redesign a conventional adder into a variable-latency approximate adder with small area overhead and well-controlled loss of computational accuracy.
- ***Achievement of almost the same quality of results as accurate addition on real error-tolerant applications*:** Instead of suppressing the occurrence frequency of erroneous computations (as in most previous studies about approximate addition), we control the magnitude of errors to fit applications with different degrees of error tolerance. In the machine learning experiment, our proposed adder outperforms other state-of-the-art approximate adders.

II. Preliminaries

In this section, we will introduce the concept of variable-latency design paradigm and the mechanism of transition detection, on which our approximate adder is based.

A. Variable-Latency Design

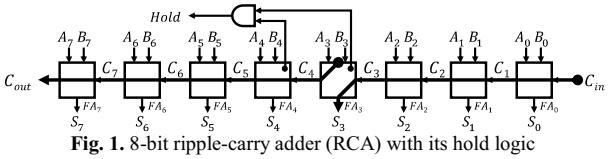


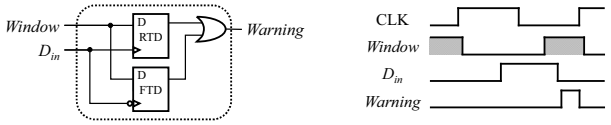
Fig. 1. 8-bit ripple-carry adder (RCA) with its hold logic

Variable-latency design (VLD) style is presented for increasing the throughput of digital circuits [17]-[21]. Instead of defining the clock period by the worst-case delay, a VLD circuit operates with a shorter clock period, T_c; and the paths are categorized into 1-cycle paths and 2-cycle paths: a 1-cycle path is a path whose delay is smaller than or equal to T_c, while a 2-cycle path is a path whose delay is larger than T_c. If no 2-cycle path is activated, the current computation can be completed within one clock cycle; otherwise, it needs two clock cycles. To distinguish between 1-cycle and 2-cycle computations, an additional circuitry called "hold logic" is constructed. The hold logic generates a hold signal that is asserted when a computation requires two clock cycles. If the clock period of a VLD circuit is derived carefully, the 2-cycle paths will be activated infrequently and the overall performance can be significantly improved by reducing the average clock period:

$$T_{avg} = P_{2-cycle} \times 2 \times T_c + \left(1 - P_{2-cycle}\right) \times T_c \qquad \textbf{(1)}$$

where $P_{2\text{-}cycle}$ is the probability of using two clock cycles.

Fig. 1 shows the variable-latency version of an 8-bit ripple-carry adder (RCA). Assuming that the delay of each 1-bit full adder (*FA*) is 1, the hold logic can be designed as $Hold = (A_3 \oplus B_3) \cdot (A_4 \oplus B_4)$ when the clock period is set to 5. For more details about hold logic construction, please refer to [19][21]. Suppose the probability of each A_i and B_i being '1' is 50%; the probability of *Hold* being '1' is 25%. Hence, the average throughput of the variable-latency RCA is $1/T_{avg}$ = 1/(0.25×10+0.75×5) = 0.16, which is higher than the throughput of the original RCA (i.e., 1/8 = 0.125), fixed-latency design with T_c equal to the total delay of the eight *FA*s.



(a) Schematic diagram **(b)** Timing diagram

Fig. 2. Transition detector

Although VLD can enhance the performance of a circuit, conventional hold logic may cause a problem called false positive. *False positive* in the context of VLD is a situation where a computation can be completed within one clock cycle but (falsely) recognized as a 2-cycle computation (positive reaction from the hold logic, i.e., *Hold* being asserted). Consider the variable-latency adder in Fig. 1 and A = $(11111111)_2$ and B = $(00000100)_2$ as its current input operands. In this case, the *Hold* signal will be asserted and thus two clock cycles will be allocated for this computation. However, if carry signal C_3 remaining from the previous clock cycle is '1', the current inputs will not activate any path with delay longer than one clock cycle. The computation requires only one cycle and the assertion of *Hold* is actually a false positive. This is because conventional hold logic simply considers static timing behavior and is unaware of dynamic activity of signal transitions, thereby significantly degrading the benefit from VLD. The false-positive rate of the adder in Fig. 1 is 0.2.

B. In-Situ Sensor: Transition Detector

A transition detector (TD) is a sensor used for monitoring the online (dynamic) transition activity in logic circuits. It can identify whether there is a transition during the specified window of time. TD was originally presented to observe the aging phenomenon [22] and timing failures [23]. Fig. 2(a) depicts the schematic diagram of a TD and an exemplary timing diagram is shown in Fig. 2(b). The TD consists of an OR gate driven by two edge-triggered D flip-flops; one (RTD) is rising-edge-triggered and the other (FTD) is falling-edge-triggered. The clock ports of the two flip-flops are connected to the monitored signal. A signal called *Window* is synchronized and generated according to the CLK signal, and fed to the data-input ports of RTD and FTD to control the detection duration. When *Window* is high, the TD will capture any transitions including both rising or falling transitions and assert its output *Warning* signal (to logic '1'), indicating the detection of a transition within the detection window. In Fig. 2(b), the monitored signal, D_{in}, has two transitions, rising before the detection window and falling within the detection window. Therefore, the TD will output '0' after the first transition and output '1' after the second transition; the transition within the detection window is detected.

Definition 1 (*late transition*): A transition is called a *late transition* if the occurrence of the transition is later than the

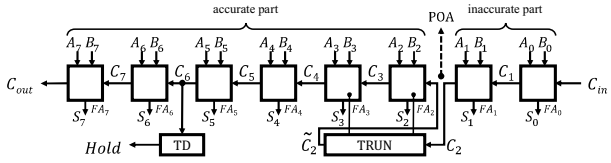


Fig. 3. Architectural overview of our 8-bit carry truncate adder (CTA)

specified threshold and does not have enough time to propagate to primary outputs within the current clock cycle, resulting in a timing error.

In this paper, we extend the use of TDs to detect late transitions by specifying *Window* properly, i.e., asserting *Window* sufficiently late in a clock cycle and de-asserting *Window* at the end of a clock cycle. Consider the same example in Fig. 2(b): the first transition on D_{in} is not a late transition while the second one is a late transition. *Warning* is accordingly asserted with the second transition, to notify that a late transition is recognized and a timing error may occur due to incomplete propagation of the late transition. In the context of VLD, the subsequent clock cycle for complete propagation will be required to the rescue.

III. Design Methodology of Our Proposed Carry Truncate Adder (CTA)

In this section, we will give an illustrative example to introduce our proposed approximate adder, Carry Truncate Adder (CTA), based on transition detection and carry truncation. Even though some applications are inherently tolerant to errors, the applied approximate adders should still maintain the results to a certain extent of quality/accuracy. Therefore, according to the accuracy requirement of the target application, we set the *point of approximation* in CTA.

Definition 2 (*point of approximation, POA*): The POA of CTA is the cut-point which divides CTA into an accurate part and an inaccurate part, where the inaccurate part contains only a small portion of less significant bits in the entire adder. The POA determines the maximum numerical difference between the correct and approximate results (a.k.a. error magnitude), which quantifies the quality of approximate addition from the accuracy perspective.

Since errors only occur at or below the POA, the results of addition may not be significantly affected by the errors. With this concept of POA, the adder can be accelerated and applied in error-tolerant applications with marginal loss of numerical accuracy.

Fig. 3 shows the architectural overview of an 8-bit CTA. Let the POA be set between FA_1 and FA_2, meaning that the "carry" from the 2-bit inaccurate part (i.e., FA_1 or FA_0) can be ignored conditionally and the optimal effective clock period for CTA is basically determined by the 6-bit accurate part. The main idea of CTA is to transform a conventional fixed-latency adder into a variable-latency design, by utilizing "TD" as the sensor-based hold logic. The logic for carry truncation (denoted by "TRUN") is then deployed to conditionally cut off carry propagations at the POA and control the error magnitude of CTA. The details about "TD" and "TRUN" will be explained in Section III.A and Section III.B, respectively. By combining these two mechanisms, our proposed approximate adder can not only achieve high performance in terms of throughout, but also guarantee high accuracy, high quality of results.

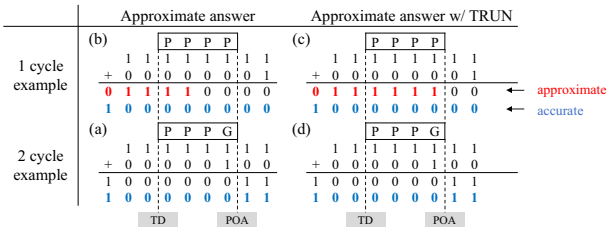


Fig. 4. Examples of CTA addition

Before detailing the proposed adder design, some variables and terms of an adder are described as follows. A_i, B_i, and C_i represent two input operands and carry-in signal feeding FA_i (the full adder at bit i), which produces S_i and C_{i+1} as the sum and carry-out, respectively. The status of each FA_i is defined as:

$$Status(FA_i) = \begin{cases} \mathbf{G}, if\ (A_i = 1\ and\ B_i = 1), \\ \mathbf{P}, if\ (A_i = 1\ and\ B_i = 0)\ or\ (A_i = 0\ and\ B_i = 1), \\ \mathbf{K}, if\ (A_i = 0\ and\ B_i = 0) \end{cases} \quad \textbf{(2)}$$

Only when $Status(FA_i) = \mathbf{P}$ (propagate), carry-out C_{i+1} has to wait for carry-in C_i; otherwise, C_{i+1} can be locally determined by input operands of FA_i. When $Status(FA_i) = \mathbf{G}$ (generate), carry-out $C_{i+1} = 1$; on the contrary, when $Status(FA_i) = \mathbf{K}$ (kill), carry-out $C_{i+1} = 0$. Besides, **X** denotes *don't care*. Suppose the probability of each A_i and B_i being '1' is 50%; the probability of $Status(FA_i)$ being **G**, **P**, and **K** is 25%, 50%, and 25%, respectively.

A. Novel Utilization of Transition Detector

After setting POA according to the accuracy requirement of a specific application, we adopt the concept of VLD paradigm to further accelerate the accurate part of CTA. To transform the conventional adder into a VLD, we use a single TD as the hold logic for determining whether or not to allow the adder a second (the subsequent) clock cycle for completing prolonged carry propagation. The goal of the TD is to detect late transitions, which are the transitions of carry signals potentially unable to propagate to C_{out} within one clock cycle. By employing such a TD as our transition-aware hold logic, online (dynamic) transition activity of CTA can be monitored. Compared to the traditional VLD whose hold logic asserting its *Hold* signal according to a single input combination (as shown in Section II.A), the false-positive rate of TD-based VLD is significantly decreased. The TD-based design will use an extra, second cycle only when a late transition really occurs, where "late" and "transition" are both necessary conditions for being detected by the TD. Therefore, it can lower the 2-cycle probability (i.e., probability of using extra, second cycles) and achieve higher performance in terms of throughput.

Consider the example in Fig. 3. The input (D_{in}) to TD is the carry-out from FA_5; that is, signal C_6 is monitored, and the longest 1-cycle path begins from FA_2 through FA_5 to TD given that errors occurring at or below the POA are considered insignificant and can be ignored. Suppose that the delay of each *FA* is 1 (arbitrary unit, a.u.) and the delay of TD is 0.5 (a.u.); the clock period of this adder can be set to 4.5 (a.u.). If the input operands of the adder are A = $(11110000)_2$ and B = $(00010000)_2$, the carry generated by FA_4 can be propagated to C_{out} with delay = 4 within one clock cycle.

However, the propagation delay from FA_2 or FA_3 to C_{out} is longer than the clock period. For example, when the input operands of the adder are A = $(11111111)_2$ and B = $(00000100)_2$

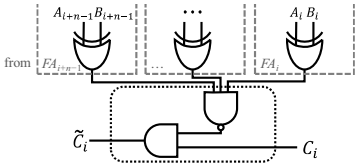


Fig. 5. Implementation of the TRUN logic

(see Fig. 4(a)), the delay of the carry generated by FA_2 and propagated to C_{out} is 6 (a.u.). To complete the carry propagation and obtain the correct result of addition (i.e., the blue number in Fig. 4(a)), an extra clock cycle is needed. Toward this end, we have to assert the detection window (*Window*) of TD to '1' from 2.5 (a.u.) to 4.5 (a.u.), with respect to each clock cycle of 4.5 (a.u.), such that TD can capture the transitions originating from FA_2 or FA_3 (i.e., late transitions) and recognize 2-cycle computations.

In general, when {*Status*(FA_5), *Status*(FA_4), *Status*(FA_3), *Status*(FA_2)} = {**P**, **P**, **G**, **X**} or {**P**, **P**, **P**, **G**} occurs, another clock cycle is required. The 2-cycle probability is $(1/2)^2\times(1/4)+(1/2)^3\times(1/4) = 3/32$. Hence, the average throughput can be further improved to $1/(3/32\times9+29/32\times4.5) \cong 0.2$. Besides, the false positive of the adder in Fig. 3 only occurs when a late transition occurs but the transition does not need one more clock cycle for propagation, i.e., {*Status*(FA_6), *Status*(FA_5), *Status*(FA_4), *Status*(FA_3)} = {**G**, **P**, **P**, **G**} or {**K**, **P**, **P**, **G**}. Therefore, by utilizing "TD" as our sensor-based hold logic, the false-positive rate can be significantly reduced to $1/32 \cong 0.03$.

After describing the addition in the accurate part, the issue about carry propagation across the POA should be handled. The carry-out from the inaccurate part needs to be truncated especially when the accurate part needs only one clock cycle to operate; otherwise, the carry-out from the inaccurate part may cause an error in the accurate part of CTA, which is not acceptable. As the example shown in Fig. 4(b), if the input operands are A = $(11111111)_2$ and B = $(00000001)_2$, a carry will be generated by FA_0 and propagated to FA_4 at the end of a clock cycle. Because of {*Status*(FA_5), *Status*(FA_4), *Status*(FA_3), *Status*(FA_2)} = {**P**, **P**, **P**, **P**}, TD will not detect any late transition in the detection window and *Warning* will not be asserted, disallowing our adder from using an extra cycle for continuous carry propagation. The result of addition turns out to be $(011110000)_2$ (i.e., the red number in Fig. 4(b)) with an error magnitude of $|(100000000)_2 - (011110000)_2| = 2^4$ due to incomplete carry propagation across the POA.

B. Deployment of Truncate Logic

To avoid incomplete carry propagation across the POA and causing unacceptable errors in the accurate part, a simple carry truncate logic called TRUN is appended to conditionally cut off the carry propagation. Fig. 5 shows the generic form of TRUN. The inputs to TRUN are C_i (carry-out from the inaccurate part) and $A_i \oplus B_i$, …, $A_{i+n-1} \oplus B_{i+n-1}$ (XOR-ed results from FA_i, …, FA_{i+n-1}). The result of $A_i \oplus B_i$ indicates whether the corresponding FA_i needs to wait for the carry from FA_{i-1} before propagating the carry to FA_{i+1}. When $A_i \oplus B_i$, …, $A_{i+n-1} \oplus B_{i+n-1}$ are all true, TRUN will stop C_i and directly assign '0' as the new carry-in (i.e., $\tilde{C}_i$) to the accurate part. This effectively avoids causing large errors due to incomplete propagation of C_i within one clock cycle, across the POA. After deploying the TRUN logic in CTA, we ensure that no error occurs beyond the

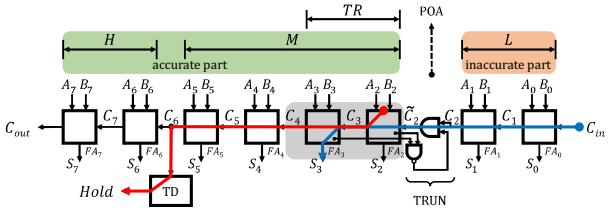


Fig. 6. CTA composition and two possible longest 1-cycle paths

position of POA. Typically, the accuracy requirement can be satisfied by feeding TRUN with the XOR-ed results from only a few *FA*s. The delay penalty incurred by TRUN includes (*i*) the delay of an AND gate and (*ii*) the delay due to extra output loading of those XOR gates in associated *FA*s. Note that the first quantity is small as compared to the delay of the entire adder, while the second quantity is basically not imposed on the delay of the critical path (longest carry chain).

In Fig. 3, we append the TRUN logic whose inputs are the XOR-ed results within FA_3 and FA_2, i.e., $n = 2$. As the example shown in Fig. 4(c) where the input operands are the same as those in Fig. 4(b), TRUN will directly assign '0' to $\tilde{C}_2$ because $(A_3 \oplus B_3)$ and $(A_2 \oplus B_2)$ are both true. Therefore, the result of addition turns out to be $(011111100)_2$ (i.e., the red number in Fig. 4(c)) with an error magnitude of $|(100000000)_2 - (011111100)_2| = 2^2$, conforming to the POA setting. Fig. 4(d) shows a final example with A = $(11111111)_2$ and B = $(00000100)_2$, and CTA produces a correct result.

IV. Performance and Accuracy Analysis

To analyze CTA in detail, we use the notations illustrated in Fig. 6. *L*, *M*, and *H* are defined to represent the sub-adders from LSB (least significant bit) to POA, from POA to TD, and from TD to MSB (most significant bit), respectively. Sub-adder *L* constitutes the inaccurate part, while sub-adders *M* and *H* constitute the accurate part (i.e., from POA to MSB). In addition, *TR* represents the sub-adders connecting their XOR-ed results to TRUN. *Delay*(•) and *Bit*(•) denote the delay and bit-width of each component, respectively. In the following discussion, the probability of each primary input (i.e., A_i and B_i) being '1' is 50%.

A. Performance Analysis (T_c and 2-cycle Probability)

The performance of CTA is determined by both clock period and 2-cycle probability as mentioned in Equation (1). The 2-cycle probability can be evaluated as:

$$P_{2\text{-}cycle} \approx \left(\frac{1}{2}\right)^{\left\lfloor \frac{T_c - Delay(H)}{Delay(FA)} \right\rfloor} \left(\frac{1}{4}\right) \times \sum_{i=1}^{\left\lfloor \frac{Delay(H)+Delay(M)-T_c}{Delay(FA)} \right\rfloor} \left(\frac{1}{2}\right)^{i-1} \quad \textbf{(3)}$$

Only when a late transition occurs, the computation will use a second clock cycle. In Fig. 6, the gray region indicates the *FA*s which may launch late transitions. When a carry-out C_i is asserted within the gray region, and $A_i \oplus B_i$, …, $A_5 \oplus B_5$ are all true (i.e., the transition can be propagated to TD), two clock cycles are allocated for this computation. For example, when {*Status*(FA_5), *Status*(FA_4), *Status*(FA_3), *Status*(FA_2)} = {**P**, **P**, **G**, **X**} or {**P**, **P**, **P**, **G**} occurs, it will be a 2-cycle computation, and the probability is 3/32.

The clock period of CTA depends on the longest 1-cycle path delay:

Table I. Comparison of 32-bit adders

	Area (μm^2)	T_c (ns)	Throughput	Energy (fJ)
RCA	165.21	5.66	0.177	98.418
CLA	771.76	0.88	1.136	310.757
VLD	202.09	3.13	0.301	115.634
ETAI	169.29	3.83	0.261	83.142
CCB	174.86	4.13	0.242	103.017
ACA	237.67	3.24	0.309	120.398
CTA	203.74	2.56	0.384	103.457

$$T_c = Max\begin{pmatrix} Delay(M) + Delay(\text{TD}), \\ Delay(L) + Delay(\text{TRUN}) + Delay(TR) \end{pmatrix} \quad \textbf{(4)}$$

As shown in Fig. 6, T_c is determined by the longer path between POA to TD (the red path) and FA_0 to the FA whose XOR-ed result is the furthermost input of TRUN (the blue path). Because the accuracy requirements of many error-tolerant applications (especially those of deep-learning applications) are strict, the blue path is seldom the longest 1-cycle path. Thus, T_c is usually determined by the placement of TD. To detect late transitions, *Window* is asserted to '1' at delay = (T_c – *Delay*(*H*)) in each cycle because the remaining cycle time is less than *Delay*(*H*), which is not enough to propagate the transition to C_{out}.

B. Accuracy Analysis (Error Magnitude, Error Frequency and Average Relative Error)

According to the accuracy requirement of a given error-tolerant application, POA is set to divide CTA into two part: the accurate part (i.e., the part from POA to MSB), operates without error occurrence; the inaccurate part (i.e., the part from LSB to POA), whose carry-out is truncated conditionally by TRUN and causes insignificant accuracy loss with error magnitude being $2^{Bit(L)}$.

The error occurs when the carry-out from the inaccurate part is asserted but truncated by TRUN. Therefore, the error frequency can be calculated as:

$$\varepsilon_{fre} = \left(\frac{1}{2}\right)^{Bit(TR)}\left(\frac{1}{4}\right) \times \sum_{i=1}^{Bit(L)}\left(\frac{1}{2}\right)^{i-1} \quad \textbf{(5)}$$

Accordingly, the situation of accuracy loss of the CTA in Fig. 6 is {*Status*(FA_3), *Status*(FA_2), *Status*(FA_1), *Status*(FA_0)} = {**P**, **P**, **G**, **X**} or {**P**, **P**, **P**, **G**}, and thus its error frequency is 3/32 ≅ 0.09.

Relative error, another common metric for estimating the quality of approximate adders, is defined as the error magnitude divided by the accurate addition result. The relative error can be calculated as:

$$\varepsilon_{rel} = \frac{|result_{correct} - result_{approx}|}{|result_{correct}|} = \frac{2^{Bit(L)}}{|result_{correct}|} \quad \textbf{(6)}$$

where $result_{correct}$ is the correct result and $result_{approx}$ is the approximate result. The average relative error of CTA in Fig. 6 is 0.0015. By utilizing both TD and TRUN, CTA can produce approximate results with high performance and low accuracy loss.

V. Experimental Results

In this section, we compare our proposed adder design against several state-of-the-art approximate adders and accurate adders. We chose three types of accurate adders including RCA, CLA, and RCA with VLD, and three approximate adders from each of three different types of approximate adders, including ETAI [10], CCB [15], and ACA [11] without its error recovery mechanism (when ACA applies its error detection and recovery mechanism, its output will be the same as accurate adders). First, we basically compare their throughput, area overhead, energy consumption and accuracy loss. To further compare these approximate adders, two error-tolerant applications are chosen to evaluate the quality of results by comparing the accuracy loss due to approximation. After demonstrating these experimental results, we will give a discussion over various approximate adders in terms of accuracy loss.

Table II. Accuracy comparison of approximate adders

	Error magnitude (normalized)	Error frequency	Relative error (average)
ETAI	2^{-22}	94.37%	1.05E-06
CCB	2^{-22}	24.98%	1.37E-06
ACA	2^{-8}	0.39%	1.14E-04
CTA	2^{-22}	12.49%	7.11E-07

A. Basic Comparisons

In this experiment, we implemented our proposed adder, three accurate adders and three approximate adders in Verilog. All the adders are synthesized by *Synopsys Design Compiler* (DC) with Synopsys SAED EDK 28nm standard cell library. For more accurate estimation of energy consumption, we perform gate-level simulations with 10^5 sets of random inputs using *Cadence NC-Sim*. Moreover, to compare the accuracy loss among these approximate adders, 10^8 random input combinations are simulated by C++.

Table I demonstrates the area, minimum clock period, average throughput (i.e., $1/T_{avg}$, where T_{avg} can be calculated by Equation (1)), and energy consumption of each adder. The width of these adders is 32 bits and the approximate adders are configured with the same maximum normalized error magnitude of 2^{-22} (normalized to the maximum magnitude of the output of a 32-bit accurate adder), except for ACA whose maximum normalized error magnitude is 2^{-8} due to its structural limitation. According to the results, although ETAI and CCB have smaller area, they can only enhance performance to be about 1.37~1.47X of RCA. ACA achieves higher performance, but it requires larger area and its error magnitude is beyond the requirements for error-tolerant applications (as demonstrated in Section V.B and Section V.C). With limited energy consumption, CTA has the best throughput among all approximate adders. The throughput of CTA is enhanced by 2.17X as compared to RCA.

Table II shows the maximum normalized error magnitude, error frequency and average relative error of each approximate adder. The error frequency of ACA is the lowest, and the one of ETAI is the highest. However, the normalized error magnitude of ACA is 2^{-8}, and thus its relative error is the highest. The approximate adders with the same design concepts as ACA focus on their error frequency and apply error recovery mechanism to eliminate errors. Because ETAI produces results with various accuracy losses in the range of [2^{-32}, 2^{-22}], the relative error of ETAI and CCB are similar. With the deployment of truncate logic, CTA can achieve the best error frequency and relative error in this experiment.

	PSNR (dB)
Accurate	31.08
CTA	27.58
CCB	23.54
ACA	23.72
ETAI	19.56

(a) (b) (c)

Fig. 7. JPEG compression result (a) PSNR of accurate addition, CTA, CCB, ACA w/o error recovery, and ETAI; (b) JPEG image using accurate addition; (c) JPEG image using CTA

B. Image Compression

JPEG is one of the most commonly used image compression method. It can effectively trade image quality for smaller image storage or less transmission cost by reducing partial high-frequency information, which is usually ignored by human vision. To test different approximate adders in JPEG application, we replace the addition associated with high-frequency signal computation.

Fig. 7 shows JPEG compression result using different adders with 50% compression ratio. The output quality of the image is measured by peak signal-to-noise ratio (PSNR). According to the results, using approximate adders can produce results with PSNR near or even higher than 20dB. Moreover, using CTA can produce similar results with the compression image using accurate addition. These results suggest that CTA can be employed in the image processing applications with only small losses in image quality.

C. Handwritten Digit Recognition

Handwritten digit recognition is a machine learning application, which trains a model to recognize numerical digits on the images. MNIST is used as the dataset and the following two softmax regression models are selected to test the accuracy loss by applying different adders. The first consists of a single-layer neural network, which can achieve 92.12% accuracy; the second consists of a multilayer convolutional neural network (CNN), which can achieve 96.15% accuracy. We train both models using accurate addition, and replace the accurate addition with approximate addition when evaluating the models.

Table III tabulates the accuracy results by applying different approximate adders. When we replace accurate addition with approximate addition at the testing stage, using CTA can achieve (almost) the same accuracy as accurate addition does in both models. Besides, if we both train and evaluate the models with CTA, the approximation-induced accuracy loss can be completely compensated. Using CCB may cause some meaningful loss of accuracy in the first model despite a competitive result in the second model. If ACA or ETAI is used, greater loss of accuracy can be observed in both models.

VI. Conclusion

In this paper, we propose a carry truncate adder (CTA) that can efficiently produce an approximate addition result with high accuracy. By simultaneously considering performance, error magnitude, and energy consumption, an in-situ sensor and a truncate logic are deployed, and thus the throughput of CTA can be enhanced to be 2.17X of RCA. The experimental results show that CTA has almost the same accuracy as exact adder and the approximation-induced error is much better than other approximate adders. Moreover, we analyze the reasons of accuracy loss among different approximate adders and address the importance of error magnitude consideration when applying approximate addition to error-tolerant applications.

Table III. Accuracy results for handwritten digit recognition on MNIST

	Single-layer network	Multilayer CNN
Accurate	92.12%	96.15%
CTA	92.12%	96.06%
CCB	86.21%	95.58%
ACA	81.32%	85.70%
ETAI	66.28%	66.32%

References

[1] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484-489, Jan. 2016.

[2] S. Venkataramani *et al.*, "Approximate computing and the quest for computing efficiency," in *Proc. of DAC*, pp. 120:1-120:6, June 2015.

[3] A. B. Kahng *et al.*, "Slack redistribution for graceful degradation under voltage overscaling," in *Proc. of ASP-DAC*, pp. 825-831, Jan. 2010.

[4] D. Mohapatra *et al.*, "Design of voltage-scalable metafunctions for approximate computing," in *Proc. of DATE*, pp. 1-6, March 2011.

[5] M. R. Choudhury and K Mohanram, "Approximate logic circuits for low overhead, non-intrusive concurrent error detection," in *Proc. of DATE*, pp. 903-908, March 2008.

[6] D. Shin and S. K. Gupta, "Approximate logic synthesis for error tolerant applications," in *Proc. of DATE*, pp.957-960, March 2010.

[7] S. Venkataramani *et al.*, "Salsa: systematic logic synthesis of approximate circuits," in *Proc. of DAC*, pp. 796-801, June 2012.

[8] J. Miao, A. Gerstlauer, and M. Orshansky, "Multi-level approximate logic synthesis under general error constraints," in *Proc. of ICCAD*, pp. 504-510, Nov. 2014.

[9] V. Gupta *et al.*, "IMPACT: imprecise adders for low-power approximate computing," in *Proc. of ISLPED*, pp.409-414, Aug. 2011.

[10] N. Zhu *et al.*, "Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing," *IEEE TVLSI*, vol. 18, no.8, pp. 1225-1229, Aug. 2010.

[11] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Proc. of DAC*, pp. 820-825, June 2012.

[12] N. Zhu, W. L. Goh, and K. S. Yeo, "An enhanced low-power high-speed adder for error-tolerant application," in *Proc. of Int'l Symp. on Integrated Circuits* (ISIC), pp. 69-72, Dec. 2009.

[13] A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: a new paradigm for arithmetic circuit design," in *Proc. of DATE*, pp. 1250-1255, March 2008.

[14] K. Du, P. Varman, and K. Mohanram, "High performance reliable variable latency carry select addition," in *Proc. of DATE*, pp. 1257-1262, March 2012.

[15] V. Camus, J. Schlachter, and C. Enz, "A low-power carry cut-back approximate adder with fixed-point implementation and floating-point precision," in *Proc. of DAC*, pp. 127:1-127:6, June 2016.

[16] M. A. Hanif *et al.*, "QuAd: design and analysis of quality-area optimal low-latency approximate adders," in *Proc. of DAC*, pp. 42:1-42:6, June 2017.

[17] L. Benini *et al.*, "Telescopic units: a new paradigm for performance optimization of VLSI designs," *IEEE TCAD*, vol. 17, no. 3, pp. 220-232, March 1998.

[18] L. Benini *et al.*, "Automatic synthesis of large telescopic units based on near-minimum timed supersetting," *IEEE Trans. on Computers*, vol. 48, no. 8, pp. 769-779, Aug. 1999.

[19] Y.-S. Su *et al.*, "An efficient mechanism for performance optimization of variable-latency designs," in *Proc. of DAC*, pp. 976-981, June 2007.

[20] D. Baneres, J. Cortadella, and M. Kishinevsky, "Variable-latency design by function speculation," in *Proc. of DATE*, pp. 1704-1709, April 2009.

[21] Y.-S. Su *et al.*, "Performance optimization using variable-latency design style," *IEEE TVLSI*, vol. 19, no. 10, pp. 1874-1883, Oct. 2011.

[22] A. Amouri and M. Tahoori, "A low-cost sensor for aging and late transitions detection in modern FPGAs," in *Proc. of Int'l Conf. on Field Programmable Logic and Applications* (FPL), pp. 329-335, Sep. 2011.

[23] L. Lai *et al.*, "SlackProbe: a low overhead in situ on-line timing slack monitoring methodology," in *Proc. of DATE*, pp. 282-287, March 2013.