Analysis and Design of Adders for Approximate Computing

SUNIL DUTT, SUKUMAR NANDI, and GAURAV TRIVEDI, Indian Institute of Technology Guwahati

The concept of approximate computing, that is, to sacrifice computation quality for computation efforts, has recently emerged as a promising design approach. Over the past decade, several research works have explored approximate computing at both the software level and hardware level of abstraction with encouraging results. At the hardware level of abstraction, adders (being the fundamental and most widely used data operators in digital systems) have attracted a significant attention for approximation. In this article, we first explain briefly the need/significance of approximate adders. We then propose four *Approximate Full Adders* (AFAs) for high-performance energy-efficient approximate computing. The key design objective behind the proposed AFAs is to curtail the length of carry propagation subjected to minimal error rate. Next, we exploit one of the proposed AFAs (optimal one) to construct an N-bit approximate adder that hereinafter is referred as "ApproxADD." An emergent property of ApproxADD is that carries do not propagate in it, and, consequently, it provides bit-width-aware constant delay ($O(1)$). ApproxADD also provides improvement in dynamic power consumption by 46.31% and in area by 28.57% w.r.t. *Ripple Carry Adder* (RCA), which exhibits the lowest power and area. Although ApproxADD provides a significant improvement in delay, power, and area, it may not be preferred for some of the error-resilient applications because its: (i) *Error Distance* (ED) is too high; and (ii) *Error Rate* (ER) increases rapidly with bit-width (N). To improve ED and ER, we exploit the concept of *carry-lifetime* and *Error Detection and Correction* logic, respectively. In this way, we introduce two more (improved) versions of ApproxADD–ApproxADD$v1$ and ApproxADD. We call these as ApproxADD$v1$ and ApproxADD$v2$ with existing approximate adders based on conventional design metrics and approximate computing design metrics. Furthermore, to inspect effectiveness of the proposed approach in real-life applications, we demonstrate image compression and decompression by replacing the conventional addition operations in *Discrete Cosine Transform* (DCT) and *Inverse Discrete Cosine Transform* (IDCT) modules with ApproxADD$v2$.

CCS Concepts: • **Hardware → Arithmetic and datapath circuits**; *Application specific integrated circuits*; *Datapath optimization*; *Emerging architectures*;

Additional Key Words and Phrases: Approximate computing, approximate adders, error-resilient applications, delay-power-area-accuracy trade-off

ACM Reference format:
Sunil Dutt, Sukumar Nandi, and Gaurav Trivedi. 2017. Analysis and Design of Adders for Approximate Computing. *ACM Trans. Embed. Comput. Syst.* 17, 2, Article 40 (December 2017), 28 pages.
https://doi.org/10.1145/3131274

This article is an extended and full version of our previously published conference article (Dutt et al. 2016b).

Authors' addresses: S. Dutt, Department of Electronics & Electrical Engineering, Indian Institute of Technology Guwahati, Guwahati, Assam, India 781039; S. Nandi, Department of Computer Science & Engineering, Indian Institute of Technology Guwahati, Guwahati, Assam, India 781039; G. Trivedi, Department of Electronics & Electrical Engineering, Indian Institute of Technology Guwahati, Guwahati, Assam, India 781039; emails: {sunil.dutt, sukumar, and trivedi}@iitg.ernet.in.

1 INTRODUCTION

With the end of Dennard scaling (around 2004–2006), increase in clock frequency and reduction in supply voltage have greatly slowed down. Under such conditions, CMOS technology can follow Moore's law, but power density increases to a level whereby processors can be as hot as a rocket nozzle (Rabaey 2009). Therefore, in mid-2000s, the microprocessor industry has shifted to multicore scaling, that is, increasing the number of cores per die per generation. In the multicore era, while the increase in transistor count continues at Moore's rate, performance and energy scaling in a cost-effective manner are now becoming difficult. From application perspective, performance gain with increased core counts is diminishing due to serial computation, synchronization, global communication, and so on (Gupta et al. 1997). From a technology perspective, more cores are only possible if the cores are slower or less utilized with each generation.[1] All together, we are now in a phase where CMOS technology scaling is becoming less and less effective at improving system capability, as shown in Figure 1. Researchers are working on post-CMOS technologies for improving the system capability, but there appears to be no obvious alternate technology that can replace *End-of-Roadmap* CMOS over the next 2 to 3 decades (see Figure 1). In such a scenario, we must either accept that computing systems are good enough or look for alternate avenues to advance them without (significant) technology progress. Recent studies show that there are several promising alternate avenues that can improve system capability equivalent to 2 to 3 decades of Moore's law (Hill and Kozyrakis 2012). Approximate computing is one of them and, in recent years, has attracted a lot of interest (Mittal 2016).

Motivation for Approximate Computing: The key motivation for approximate computing is *error-resilient applications*. Error-resilience refers to the property of an application to accept outputs despite some of its underlying computations being executed using approximate software and/or hardware. Recent studies show that a very high degree of error resilience (≈83% (Chippa et al. 2013)) is prevalent in a broad spectrum of applications, including machine learning, web search, probabilistic analytics, scientific computing, image, audio and video processing, wireless communication, and so on. All these applications are very resource hungry, but are amenable to inherent error resilience. In such a scenario, approximate computing could be a good candidate in improving the system capability by trading-off computation quality (e.g., accuracy) for computation efforts (e.g., delay and power consumption).

Significance of Approximate Adders: Over the past decade, several research works have explored approximate computing at both the software level and hardware level of abstraction (Mittal 2016). Software techniques skip algorithm-level computations, whereas hardware techniques modify designs at the circuit level of abstraction. At the hardware level of abstraction, most of the work (in approximate computing paradigm) has been proposed on arithmetic units. Further, among arithmetic units, adders have attracted the strongest attention for approximation (Jiang et al. 2015; Dutt et al. 2016a). Here, we explain briefly the need/significance of approximate adders. (1) Fundamental data operator: Binary arithmetic is an essential part of digital systems. In binary arithmetic, adders are the key to perform arithmetic operations, that is, all arithmetic operations, such as addition, subtraction, multiplication, and division use adders as the key structure. Besides arithmetic operations, adders are used to perform increment, decrement, and many similar operations. Therefore, being the most widely used fundamental data operators, adders have attracted a significant interest for approximation. (2) Delay/power bottleneck: In all digital circuits, delay and power are two conflicting design parameters, that is, improvement of one usually demands the

[1]With progression of Moore's law, the percentage of a chip that can actively be used within its own power budget is dropping exponentially (Esmaeilzadeh et al. 2011). Even at 22nm, 21% of a CMOS chip must be powered off. At 8nm, this number is expected to reach up to 50%–80%.

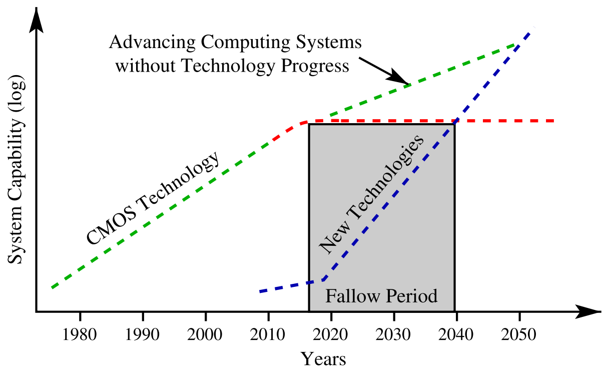


Fig. 1. Researchers are working to harvest new gains during the "fallow period" to advance computing systems without (significant) technology progress (Hill and Kozyrakis 2012).

sacrifice of the other. However, this situation is more severe in the case of adders. Since delay and power of adders increase rapidly with bit-width (N), for a marginal improvement in delay/power, we have to sacrifice an immense amount of power/delay. Consequently, at a microarchitecture level of abstraction, adders have become the key delay/power bottleneck of digital systems. One possible way to overcome this situation is to approximate adders, that is, to sacrifice accuracy for delay and/or power. (3) Parametric yield loss: The parametric yield loss (which is increasing rapidly with CMOS technology scaling) has become a matter of concern for the semiconductor industry (Miller 2005). The parametric yield loss occurs when there is an unacceptable mismatch between the intended and obtained design specifications. Adders being the basic building blocks of digital systems determine the overall parametric yield of digital systems (particularly, of digital systems designed for datapath dominated applications (CEVA 2012)). The parametric yield of such digital systems can be improved by exploiting approximate adders (Shin and Gupta 2008; Dutt et al. 2016b).

Design Approach for Approximate Adders: Considering the aforementioned motivation, several approximate adders have been proposed in the literature (Jiang et al. 2015; Dutt et al. 2016a). The key design approach for all existing approximate adders is to curtail the length of carry propagation. Here, we explain briefly the observations behind this. We know that the critical path in conventional adders depends on the length of carry propagation. In the worst-case, carry generates at LSBs and propagates to MSBs. In such conditions, the length of carry propagation is $\approx N$. But, in real-life applications, such conditions rarely happen and in most of the cases, the longest length of carry propagation is much shorter as compared to bit-width. Researchers/designers have theoretically established that in most of the cases, the longest length of carry propagation is close to $log_2(N)$ (Parhami 2010) and is less than $log_2(N) + 12$ (Verma et al. 2008). In other words, the worst-case carry propagation (i.e., carry propagates $\approx N$ bits) rarely happens. Therefore, if carry propagation is curtailed, then the re-designed adder may provide approximate results, but in return, it provides a significant improvement in delay and power consumption.

One of the most commonly used approaches to curtail the length of carry propagation is *Approximate Full Adder* (AFA). AFAs are designed by modifying conventional *Full Adders* (FAs) at the gate level (Shin and Gupta 2008) and transistor level (Gupta et al. 2013) of abstraction in such a way that *Carry-out* (C_{out}) remains independent of *Carry-in* (C_{in}). In this article, we propose four AFAs. While designing the proposed AFAs, our objective is to obtain C_{out} independent of C_{in} subjected to the AFAs provide minimal error rate. We then exploit one of the proposed AFAs (optimal one) to construct an N-bit approximate adder, which we call "ApproxADD." To improve

Error Distance (ED)[2] of ApproxADD, we perform higher-order k-bit operations accurately because impact of errors in higher-order bits is more severe as compared to the errors in lower-order bits. We call this version as "ApproxADD$v1$." For estimating the number of higher-order k-bits that should be performed accurately, we exploit the concept of *carry-lifetime*. Further, to improve *Error Rate* (ER)[3] of ApproxADD$v1$, we propose an *Error Detection and Correction* (EDC) logic that detects errors, and accordingly, corrects them. We call this version as "ApproxADD$v2$." Note that a preliminary work of this article has been reported in our previously published conference article (Dutt et al. 2016b). In Dutt et al. (2016b), we exploit the proposed approach for yield enhancement, whereas in this article, we exploit the proposed approach for high-performance energy-efficient approximate computing. Therefore, design objective, analytical models, simulation results, analysis, and so on, presented in this article are different as compared to Dutt et al. (2016b). Furthermore, the key extensions that have been incorporated in this article are summarized as follows:

(1) We provide detailed analysis and analytical/mathematical models to estimate conventional design metrics (such as delay, power, and area) and approximate computing design metrics (such as ED and ER) of ApproxADDs.
(2) We evaluate ApproxADD$v1$ and ApproxADD$v2$ over three additional state-of-the-art approximate computing design metrics that are generally used to measure the "acceptance threshold" of error-resilient applications.
(3) For the ease of illustration, we discuss the proposed approach w.r.t. *Ripple Carry Adder* (RCA). However, the proposed approach can be customized for any adder architecture. We demonstrate (using parallel-prefix architectures) how the proposed approach can be customized for other adder architectures.
(4) To inspect efficiency of the proposed approach, we compare ApproxADD$v1$ and ApproxADD$v2$ with existing approximate adders based on both conventional design metrics and approximate computing design metrics.
(5) For evaluating effectiveness of the proposed approach in real-life applications, we demonstrate approximate image compression and decompression by replacing the conventional addition operations in *Discrete Cosine Transform* (DCT) and *Inverse Discrete Cosine Transform* (IDCT) modules with ApproxADD$v2$.

The rest of the article is organized as follows. Based on the method of logical effort, Section 2 provides analytical models for estimating delay, power, and area of ApproxADDs. Section 3 presents all the four proposed AFAs and their delay, power, and area analysis. Section 4 presents ApproxADD, and Section 5 presents ApproxADD$v1$ and ApproxADD$v2$ along with their analytical results and simulation results. Section 6 provides simulation results and discussion of the proposed approach in respect of acceptance threshold, parallel-prefix adder architectures, and comparison with existing approximate adders. Section 7 evaluates effectiveness of the proposed approach in real-life applications. Finally, Section 8 concludes the article.

2 ABSTRACTING DELAY, POWER, AND AREA MODELS

In digital circuit design, there are different levels of design abstraction, such as transistor, gate, and so on. At each level of design abstraction, several delay, power, and area estimation methods have been proposed in literature. One of the most commonly used methods at gate level of

[2]ED is defined as the arithmetic distance between the correct and erroneous outputs, that is, $ED = |R_c - R_e|$, where R_c is the output of accurate adder and R_e is the output of approximate adder (Liang et al. 2013).
[3]For a set of inputs, ER is defined as the probability of ED to be non-zero, that is, $ER = \frac{1}{n}\sum_{i=1}^{n} P(ED_i \neq 0)$, where n is the input set size (Liang et al. 2013).

abstraction is *logical effort* (Sutherland et al. 1999). We also use the same method for estimating delay, power, and area of ApproxADDs. Note that the parameters described in this section are used later in this article.

2.1 Delay Estimation

According to the method of logical effort, total delay (d) incurred by a static CMOS logic gate (measured in terms of basic delay unit τ)[4] can be given by

$$d = f + p_d, \tag{1}$$

where f is the effort delay and p_d is the parasitic delay. The effort delay further consists of two components: (i) logical effort (g), which captures the effect of logic complexity; and (ii) electrical effort (h), which captures the effect of external load being driven by the logic gate. The effort delay is given by the product of these two components as

$$f = g \times h, \tag{2}$$

where logical effort is given by the ratio of input capacitance of the logic gate ($C_{in_{gate}}$) to the input capacitance of a unit standard CMOS inverter ($C_{in_{inv}}$) as

$$g = \frac{C_{in_{gate}}}{C_{in_{inv}}}, \tag{3}$$

and electrical effort is given by the ratio of load capacitance (C_L) being driven by the logic gate to the input capacitance of the logic gate ($C_{in_{gate}}$) as

$$h = \frac{C_L}{C_{in_{gate}}}. \tag{4}$$

Substituting the value of effort delay from Equation (2) into Equation (1), we obtain the basic equation that models delay[5] of a static CMOS logic gate as

$$d = (g \times h) + p_d. \tag{5}$$

We can now compute the path delay (D)[6] of an n-stage CMOS digital circuit by adding the delay (d) of individual logic gates in that path as

$$D = \sum_{i=1}^{n} d_i = \sum_{i=1}^{n} \left((g_i \times h_i) + p_{d_i} \right). \tag{6}$$

2.2 Power Estimation

The total power consumption of a CMOS logic gate mainly consists of two components: (i) static power; and (ii) dynamic power. The dynamic power consumption (p_{dyn}) which dominates the total power consumption can be given by

$$p_{dyn} = \alpha \, C_{out} \, V_{DD}^2 \, f_{clk}, \tag{7}$$

where α, C_{out}, V_{DD}, and f_{clk} are the switching activity factor, output capacitance, supply voltage, and clock frequency, respectively. The output capacitance further consists of two components:

[4]Delay of CMOS logic gates depends on process technology. To isolate the effects of a particular process technology, unless otherwise indicated, we measure all the delays in units of τ (Sutherland et al. 1999).

[5]In Sutherland et al. (1999), to study the effect of logic complexity, the authors introduce logical effort (g) and to study the effect of external load being driven by logic gates, the authors introduce electrical effort (h). Therefore, simplification of Equation (2) as $(C_{in_{gate}}/C_{in_{inv}}) \times (C_L/C_{in_{gate}}) = C_L/C_{in_{inv}}$ is not advisable.

[6]To represent delay, power, and area of logic gates, we use small letters (d, p, and a), whereas to represent delay, power, and area of digital circuits, we use capital letters (D, P, and A).

Table 1. Probability of Output Node to Be 1 of Various Logic Gates as a Function of Their Input Nodes Probability (Weste and Harris 2010)

Logic Gates		P_1
NOT	A	$1 - P_A$
2-input NAND	A and B	$1 - (P_A \times P_B)$
2-input NOR	A and B	$(1 - P_A)(1 - P_B)$
2-input XOR	A and B	$P_A(1 - P_B) + P_B(1 - P_A)$
3-input NAND	A, B and C	$1 - (P_A \times P_B \times P_C)$

(i) load capacitance (C_L); and (ii) parasitic capacitance (C_p). Substituting the value of output capacitance into Equation (7) yields

$$p_{dyn} = \alpha \, (C_L + C_p) \, V_{DD}^2 \, f_{clk}. \tag{8}$$

Estimating the dynamic power consumption using Equation (8) is very cumbersome and time-consuming. To make this easy, (Kabbani 2008) characterizes the dynamic power consumption in terms of logical effort terminologies by normalizing it w.r.t. the dynamic power consumption of a unit standard CMOS inverter as

$$p_{nm\_inv} = \frac{\alpha_{gate}}{\alpha_{inv}} \left(\frac{h \times C_{in_{gate}}}{C_{in_{inv}}} + \frac{C_{p_{gate}}}{C_{p_{inv}}} \right) = \alpha_{nm} \, (h \times g + p_p), \tag{9}$$

where p_{nm_inv} and α_{nm} are the normalized dynamic power consumption and normalized switching activity factor of the logic gate w.r.t. a unit standard CMOS inverter, respectively, and p_p is the parasitic power consumption of the logic gate. Since dynamic power consumption depends on the area (a) of the logic gate, Equation (9) is valid only for *template gates*. A template gate refers to the logic gate whose output current is equal to the output current of a unit standard CMOS inverter. For any other logic gate, the normalized dynamic power consumption can be given by

$$p_{nm\_inv} = a_{nm} \, \alpha_{nm} \, (h \times g + p_p), \tag{10}$$

where a_{nm} is the normalized area of the logic gate w.r.t. corresponding template gate. We can now compute the normalized dynamic power consumption (P_{nm_inv}) of a CMOS digital circuit that contains m logic gates by adding the normalized dynamic power consumption (p_{nm_inv}) of individual logic gates in that circuit as

$$P_{nm\_inv} = \sum_{i=1}^{m} p_{nm\_inv_i} = \sum_{i=1}^{m} \left(a_{nm_i} \, \alpha_{nm_i} \, (h_i \times g_i + p_{p_i}) \right). \tag{11}$$

While determining the normalized dynamic power consumption using Equation (11), it is necessary to compute the switching activity factor at each node. Switching activity factor of a node is the probability that it switches from 0 to 1, that is, probability of the node to be 0 on one cycle and to be 1 on the next cycle. Assuming that the probability is uncorrelated from cycle to cycle, switching activity factor of a node can be given by

$$\alpha = P_0 \times P_1 = P_1(1 - P_1), \tag{12}$$

where P_0 is the probability of the node to be 0 and P_1 is the probability of the node to be 1. Table 1 lists P_1 of output node of various logic gates used in this article, where P_A, P_B, and P_C are the probabilities of input nodes A, B, and C to be 1, respectively (Weste and Harris 2010). Assuming that inputs are uniformly distributed (i.e., P_1 of input nodes is 0.5), we first compute P_1 of

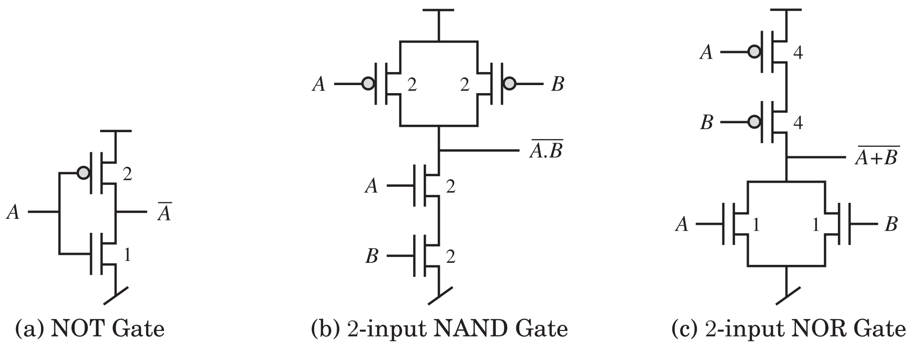


Fig. 2. Transistor-level implementation: (a) NOT gate, which has $a = 3$ units and $C_{in_{gate}} = 3$ units; (b) 2-input NAND gate, which has $a = 8$ units and $C_{in_{gate}} = 4$ units; and (c) 2-input NOR gate, which has $a = 10$ units and $C_{in_{gate}} = 5$ units. Note that we size these logic gates according to the method of logical effort.

all the circuit nodes using Table 1. We then compute switching activity factor of the circuit nodes using Equation (12).

2.3 Area Estimation

Circuit designers generally measure area in terms of gate count or transistor count. However, this type of analysis does not include the effect of transistor sizes, and therefore, their estimation accuracy is poor. For example, as shown in Figure 2, both NAND gate and NOR gate have same transistor count, but due to different transistor sizes, their actual silicon areas are different. To improve the estimation accuracy, we measure area directly in terms of transistor sizes. According to our analysis, NAND gate has area, $a = 8$ units and NOR gate has area, $a = 10$ units (see Figure 2). Similarly, we compute area of other logic gates used in this article. We then compute the area (A) of a CMOS digital circuit that contains m logic gates by adding the area (a) of individual logic gates in that circuit as: $A = \sum_{i=1}^{m} a_i$.

3 DESIGNING APPROXIMATE FULL ADDERS

Consider a FA which is the basic building block of commonly used adder architectures. An FA can be expressed by the well-known Boolean functions as

$$S = A \oplus B \oplus C_{in}, \tag{13}$$

$$C_{out} = A.C_{in} + B.C_{in} + A.B, \tag{14}$$

where A, B, and C_{in} are the binary inputs and Sum (S) and C_{out} are the binary outputs. Using Equation (13) and Equation (14), there are different ways (both at transistor level and gate level of abstraction) to implement an FA. One of the most widely used gate-level implementations of FA is shown in Figure 3. Let us now consider an N-bit RCA, which is generally constructed by cascading N FAs in series, as shown in Figure 4. An emergent property of RCA is that it exhibits the smallest silicon area. However, in RCA, since C_{out} of ith FA is feed directly to the C_{in} of $(i + 1)$th FA, it is significantly slower as compared to parallel-prefix adders (Weste and Harris 2010). Moreover, a significant portion of dynamic power consumed in an N-bit RCA is due to re-switching of internal nodes when carries propagate from LSBs to MSBs. One possible way to overcome the aforementioned limitations is to curtail the length of carry propagation. The re-designed RCA may provide approximate results, but in return, it provides a significant improvement in delay and dynamic power consumption.

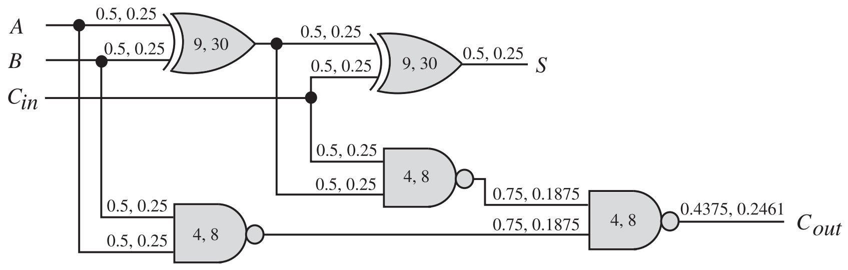


Fig. 3. Gate-level logic implementation of conventional FA. Here, each logic gate is shown with two numbers, where the first number indicates input capacitance ($C_{in_{gate}}$) of the logic gate and the second number indicates area (a) of the logic gate (Sutherland et al. 1999). Further, input and output nodes of the logic gates are also shown with two numbers, where the first number indicates P_1, that is, probability of the node to be 1 and the second number indicates switching activity factor (α) of the node (Weste and Harris 2010).

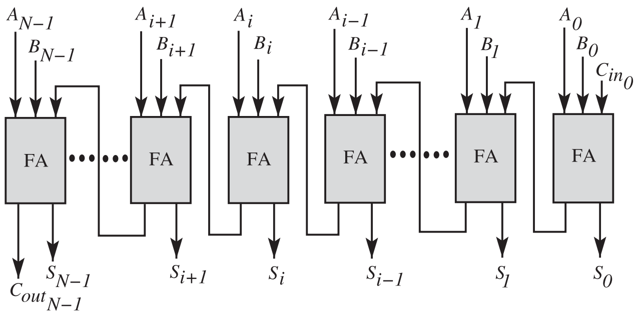


Fig. 4. Architecture of an N-bit RCA designed using N FAs in series.

3.1 Proposed Approximate Full Adders

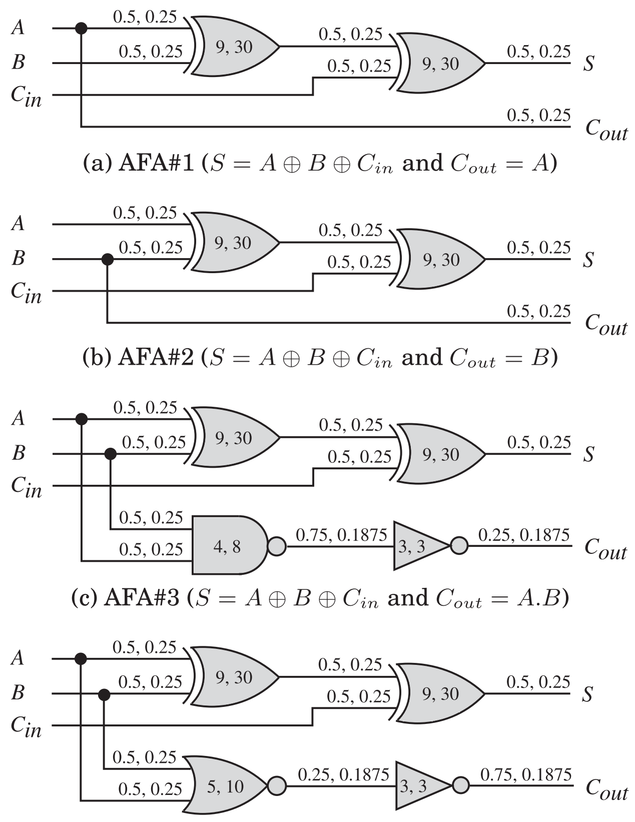
AFA is one of the commonly used approaches to curtail the length of carry propagation. AFAs are designed by modifying conventional FAs (at transistor level (Gupta et al. 2013) and gate level (Shin and Gupta 2008) of abstraction) in such a way that C_{out} is independent of C_{in}. In a FA, to secure C_{out} independent of C_{in}, C_{out} should be 0, 1, or a Boolean function of inputs A and B only. In such a scenario, there can be a total of $2^{(2^2)} = 16$ approximate functions of C_{out}. All these 16 approximate functions are tabulated in Table 2 with corresponding correct and incorrect entries. Assuming that inputs are uniformly distributed, the probability of approximate C_{out} to be correct is also mentioned in Table 2. From Table 2, we observe that

(1) In an AFA, if we consider C_{out} as 0, 1, $A \oplus B$, $A \odot B$, $\overline{A}.B$, $A.\overline{B}$, $\overline{A} + B$ or $A + \overline{B}$, then the C_{out} has correct output for four of eight input combinations, and therefore, the probability of C_{out} to be correct is $4/8 = 0.5$.
(2) In an AFA, if we consider C_{out} as $\overline{A}$, $\overline{B}$, $\overline{A.B}$ or $\overline{A+B}$, then the C_{out} has correct output for two of eight input combinations, and therefore, the probability of C_{out} to be correct is $2/8 = 0.25$.
(3) In an AFA, if we consider C_{out} as A, B, $A.B$ or $A + B$, then the C_{out} has correct output for six of eight input combinations, and therefore, the probability of C_{out} to be correct is $6/8 = 0.75$.

It is clear from the earlier observations that in an AFA, if we consider C_{out} as A, B, $A.B$, or $A + B$, then the AFA provides lower error probability. Consequently, an N-bit approximate adder

Table 2. All Possible Approximate Functions of C_{out} in Which C_{out} is Independent of C_{in}

Inputs				Approximate Functions of C_{out} in which C_{out} is Independent of C_{in}															
A	B	C_{in}	C_{out}	0	1	A	B	$\overline{A}$	$\overline{B}$	$A.B$	$A+B$	$\overline{A.B}$	$\overline{A+B}$	$A \oplus B$	$A \odot B$	$\overline{A}.B$	$A.\overline{B}$	$\overline{A}+B$	$A+\overline{B}$
0	0	0	0	0 ✓	1 ×	0 ✓	0 ✓	1 ×	1 ×	0 ✓	0 ✓	1 ×	1 ×	0 ✓	1 ×	0 ✓	0 ✓	1 ×	1 ×
0	0	1	0	0 ✓	1 ×	0 ✓	0 ✓	1 ×	1 ×	0 ✓	0 ✓	1 ×	1 ×	0 ✓	1 ×	0 ✓	0 ✓	1 ×	1 ×
0	1	0	0	0 ✓	1 ×	0 ✓	1 ×	1 ×	0 ✓	0 ✓	1 ×	1 ×	0 ✓	1 ×	0 ✓	1 ×	0 ✓	1 ×	0 ✓
0	1	1	1	0 ×	1 ✓	0 ×	1 ✓	1 ✓	0 ×	0 ×	1 ✓	1 ✓	0 ×	1 ✓	0 ×	1 ✓	0 ×	1 ✓	0 ×
1	0	0	0	0 ✓	1 ×	1 ×	0 ✓	0 ✓	1 ×	0 ✓	1 ×	1 ×	0 ✓	1 ×	0 ✓	0 ✓	1 ×	0 ✓	1 ×
1	0	1	1	0 ×	1 ✓	1 ✓	0 ×	0 ×	1 ✓	0 ×	1 ✓	1 ✓	0 ×	1 ✓	0 ×	0 ×	1 ✓	0 ×	1 ✓
1	1	0	1	0 ×	1 ✓	1 ✓	1 ✓	0 ×	0 ×	1 ✓	1 ✓	0 ×	0 ×	0 ×	1 ✓	0 ×	0 ×	1 ✓	1 ✓
1	1	1	1	0 ×	1 ✓	1 ✓	1 ✓	0 ×	0 ×	1 ✓	1 ✓	0 ×	0 ×	0 ×	1 ✓	0 ×	0 ×	1 ✓	1 ✓
Probability of C_{out} to be correct				0.5	0.5	0.75	0.75	0.25	0.25	0.75	0.75	0.25	0.25	0.5	0.5	0.5	0.5	0.5	0.5



(a) AFA#1 ($S = A \oplus B \oplus C_{in}$ and $C_{out} = A$)

(b) AFA#2 ($S = A \oplus B \oplus C_{in}$ and $C_{out} = B$)

(c) AFA#3 ($S = A \oplus B \oplus C_{in}$ and $C_{out} = A.B$)

(d) AFA#4 ($S = A \oplus B \oplus C_{in}$ and $C_{out} = A + B$)

Fig. 5. Gate-level logic implementations of the proposed AFAs. The numbers indicated on the logic gates and on the input and output nodes of the logic gates have been described earlier in Figure 3.

designed using either of these four AFAs provides higher accuracy. We call these AFAs as AFA#1, AFA#2, AFA#3, and AFA#4, respectively. The gate-level logic implementations of these AFAs are shown in Figure 5.

3.2 Delay, Power, and Area Analysis

For estimating delay, power, and area of conventional FA and the proposed AFAs, we use analytical models described in Section 2. We first compute design parameters, such as α, g, h, p_d, p_p, a, d, and p_{nm_inv} of the logic gates used in conventional FA and the proposed AFAs. Our computed values are tabulated in Table 4. Then, using Table 4, we compute delay (D), normalized dynamic power consumption (P_{nm_inv}), and area (A) of conventional FA and the proposed AFAs by adding the delay (d), normalized dynamic power consumption (p_{nm_inv}), and area (a) of individual logic gates in that circuit, respectively. For example, in conventional FA (Figure 3), output S has delay $D = 8.33 + 5 = 13.33$ and output C_{out} has delay $D = 8.33 + 3.33 + 6.33 = 17.99$, whereas in AFA#3 (Figure 5(c)), output S has delay $D = 7 + 5 = 12$ and output C_{out} has delay $D = 3 + 4 = 7$. Therefore, as compared to conventional FA, AFA#3 improves delay for output S by 9.98% and for output C_{out} by 61.09%. Further, conventional FA has normalized dynamic power consumption $P_{nm\_inv} = 9.33 + 6 + 2.49 + 2.49 + 6.23 = 26.54$ and area $A = 30 + 30 + 8 + 8 + 8 = 84$ units, whereas AFA#3 has normalized dynamic power consumption $P_{nm\_inv} = 8 + 6 + 2.25 + 3 = 19.25$

Table 3. Estimated D, P_{nm_inv}, A, and $PDAP$ of Conventional FA (Figure 3) and the Proposed AFAs (Figure 5)

Adder Architectures	D		ΔD (%)		P_{nm_inv}	ΔP_{nm_inv} (%)	A	ΔA (%)	$PDAP$	$\Delta PDAP$ (%)
	S	C_{out}	S	C_{out}						
Conventional FA	13.33	17.99	0	0	26.54	0	84	0	40.106E+3	0
Proposed AFA#1	12	0	-9.98	-100	14	-47.25	60	-28.57	10.080E+3	-74.86
Proposed AFA#2	12	0	-9.98	-100	14	-47.25	60	-28.57	10.080E+3	-74.86
Proposed AFA#3	12	7	-9.98	-61.09	19.25	-27.47	71	-15.48	16.401E+3	-59.11
Proposed AFA#4	12	7	-9.98	-61.09	19.25	-27.47	73	-13.09	16.863E+3	-57.95

Table 4. Design Parameters of the Logic Gates Used in Conventional FA (Figure 3) and the Proposed AFAs (Figure 5)

Logic Gates	α	g	h	p_d	p_p	a	d	p_{nm_inv}
INV	0.1875	1	3^*	1	1	3	4	3
NAND	0.1875	4/3	1	2	2	8	3.33	2.49
NAND	0.2461	4/3	$13/4^*$	2	2	8	6.33	6.23
NAND	0.1875	4/3	3/4	2	2	8	3	2.25
NOR	0.1875	5/3	3/5	2	2	10	3	2.25
XOR	0.25	3	13/9	4	5	30	8.33	9.33
XOR	0.25	3	$1/3^{\#}$	4	5	30	5	6
XOR	0.25	3	1	4	5	30	7	8

Here, $*$ and # signify that while computing the electrical effort (h) of output logic gates, we assume that C_{out} drives C_{in} and S drives a unit standard CMOS inverter, respectively.

and area $A = 30 + 30 + 8 + 3 = 71$ units. Therefore, as compared to conventional FA, AFA#3 improves dynamic power consumption by 27.47% and area by 15.48%. Similarly, we compute delay, normalized dynamic power consumption, and area of AFA#1, AFA#2, and AFA#4. Our computed values along with *Power-Delay-Area-Product* (PDAP) of conventional FA and the proposed AFAs are tabulated in Table 3. The percentage change in delay (ΔD), normalized dynamic power consumption (ΔP_{nm_inv}), area (ΔA), and PDAP ($\Delta PDAP$) of the proposed AFAs w.r.t. conventional FA is also mentioned in Table 3.

4 DESIGNING N-BIT APPROXIMATE ADDER

We now design an N-bit approximate adder using the proposed AFAs. Table 2 shows that all the four proposed AFAs exhibit similar error probability. Therefore, an N-bit approximate adder designed using either of these four AFAs provides similar error characteristics (such as ED and ER). However, as shown in Table 3, these AFAs have different delay, power, area, and PDAP. Among all the four proposed AFAs, AFA#1 and AFA#2 provide smaller delay, power, area, and PDAP. Therefore, while designing an N-bit approximate adder that provides optimal *Delay-Power-Area-Accuracy* (DPAA) tradeoff, one should choose either AFA#1 or AFA#2. An N-bit approximate adder (which we call "ApproxADD") designed using AFA#1 is shown in Figure 6.

4.1 Delay, Power, and Area Analysis

In an N-bit RCA, the worst-case delay is given by the delay to compute sum in the last stage and the delay to compute carry propagation in the remaining $(N - 1)$ stages. In case of an N-bit ApproxADD, since there is no carry propagation, the worst-case delay is given by the delay to compute sum in AFA#1 only. Therefore, using Table 3 and Table 4, the normalized delay (D_{nm}) of

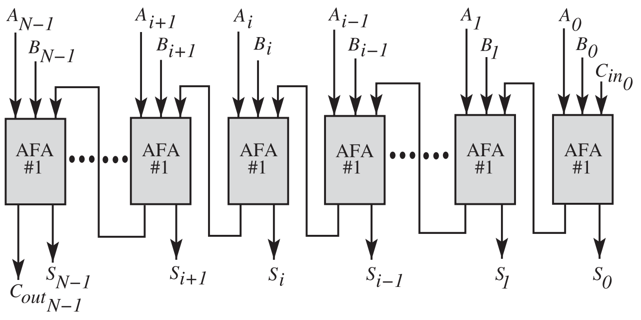


Fig. 6. Architecture of an N-bit ApproxADD designed using N AFA#1.

an N-bit ApproxADD $w.r.t.$ RCA can be given by

$$D_{nm} = \frac{D_{AFA\#1}}{D_{FA[S]} + (N-1) \times D_{FA[C_{in}:C_{out}]}} = \frac{12}{13.33 + (N-1) \times 9.66} = \frac{1.2422}{N + 0.3799}, \tag{15}$$

where $D_{AFA\#1}$ is the delay to compute sum in AFA#1, $D_{FA[S]}$ is the delay to compute sum in conventional FA, and $D_{FA[C_{in}:C_{out}]}$ is the delay to compute carry propagation ($C_{in} \rightarrow C_{out}$) in conventional FA. Further, the normalized dynamic power consumption (P_{nm}) and normalized area (A_{nm}) of an N-bit ApproxADD w.r.t. RCA can be given by

$$P_{nm} = \frac{N \times P_{nm\_inv_{AFA\#1}}}{N \times P_{nm\_inv_{FA}}} = 0.5275, \tag{16}$$

$$A_{nm} = \frac{N \times A_{AFA\#1}}{N \times A_{FA}} = 0.7143, \tag{17}$$

where $P_{nm\_inv_{FA}}$ and $P_{nm\_inv_{AFA\#1}}$ are the normalized dynamic power consumptions of conventional FA and AFA#1 w.r.t. a unit standard CMOS inverter, and A_{FA} and $A_{AFA\#1}$ are the areas of conventional FA and AFA#1, respectively.

4.2 ED and ER Analysis

As shown in Table 2, AFA#1 provides erroneous carry for two of eight input combinations. These two input combinations are $\overline{A}.B.C_{in}$ and $A.\overline{B}.\overline{C}_{in}$. Therefore, in ApproxADD, an erroneous carry generates at the ith bit position only when $\overline{A}_i.B_i.A_{i-1} = 1$ or $A_i.\overline{B}_i.\overline{A}_{i-1} = 1$ because C_{in_i} is A_{i-1} in ApproxADD. Further, we know that in arithmetic operations, erroneous carry generated at ith bit position affects the output at $(i+1)$th bit position. Accordingly, ED of an N-bit ApproxADD can be given by the weighted sum of input combinations $\overline{A}_i.B_i.A_{i-1}$ and $A_i.\overline{B}_i.\overline{A}_{i-1}$ as

$$ED \approx \sum_{i=0}^{N-1} 2^{i+1} \times \left\{ (\overline{A}_i.B_i.A_{i-1}) + (A_i.\overline{B}_i.\overline{A}_{i-1}) \right\}, \tag{18}$$

where $\cdot$ and + represent the binary AND and OR operations, respectively. Note that in ApproxADD, either of the input combinations ($\overline{A}_i.B_i.A_{i-1}$ or $A_i.\overline{B}_i.\overline{A}_{i-1}$) cannot occur consecutively. However, these input combinations can occur in an alternative fashion. This is the worst-case in which ApproxADD provides maximum error. For example, let us assume that A is 01010101, B is 10101010, and C_{in_0} is 0. In this case, $\overline{A}_i.B_i.A_{i-1}$ OR $A_i.\overline{B}_i.\overline{A}_{i-1}$ is 11111111, which implies that if we add A and

B using ApproxADD, then error will occur at all bit positions. Since accurate sum is 11111111, approximate sum (using ApproxADD) should be 00000000. But ApproxADD computes it as 01010101. This difference is because in ApproxADD, two consecutive erroneous carries generated from ith bit position do not propagate erroneous carry for $(i + 2)$th bit position (see Figure 11 in (Dutt et al. 2016b)). The reason behind this is the fact that twice the complement of a function is the function itself. In other words, an erroneous carry generated at ith bit position masks the effect of erroneous carry generated at $(i + 1)$th bit position. Therefore, Equation (18) can be corrected by subtracting the weighted sum of input combinations that generate consecutive erroneous carries as

$$\begin{aligned} ED \approx & \sum_{i=0}^{N-1} 2^{i+1} \times \left\{(\overline{A}_i.B_i.A_{i-1}) + (A_i.\overline{B}_i.\overline{A}_{i-1})\right\} \\ & - \sum_{i=0}^{N-1} 2^{i+1} \times \left\{(\overline{A}_i.B_i.A_{i-1}) \,.\, (A_{i-1}.\overline{B}_{i-1}.\overline{A}_{i-2})\right\} \\ & - \sum_{i=0}^{N-1} 2^{i+1} \times \left\{(A_i.\overline{B}_i.\overline{A}_{i-1}) \,.\, (\overline{A}_{i-1}.B_{i-1}.A_{i-2})\right\}. \end{aligned} \tag{19}$$

Note that Equation (19) is still an approximation because overlapping cases are not taken into consideration. Further, in most of the error-resilient applications, *Relative Error Distance* (RED) is more meaningful as compared to absolute ED, where RED is defined as the ED normalized to correct output (R_c) (Liang et al. 2013). For a set of n inputs, *Mean* RED (MRED) of an N-bit ApproxADD can be given by

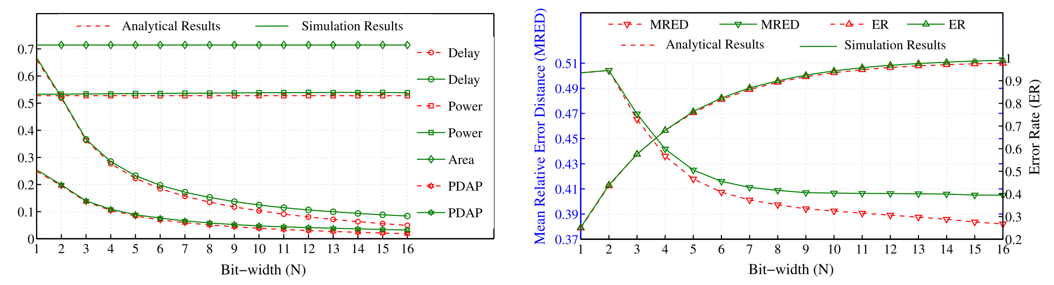
$$MRED = \frac{1}{n}\sum_{i=1}^{n} RED_i = \frac{1}{n}\sum_{i=1}^{n}\frac{ED_i}{R_{c_i}}. \tag{20}$$

Further, to model ER of ApproxADD, we use an empirical approach, that is, we model ER using simulation results. From our simulation results, we observe that ER of ApproxADD forms a *Geometric Progression* (GP) with scalar factor ≈1/4 and common ratio ≈3/4. Accordingly, ER of an N-bit ApproxADD can be given by

$$ER = \sum_{i=1}^{N}\frac{1}{4}\times\left(\frac{3}{4}\right)^{i-1} = \sum_{i=1}^{N}\frac{3^{i-1}}{4^i}. \tag{21}$$

4.3 Simulation Setup

For assessing the conventional design metrics (such as delay, power, and area), we design RCA and ApproxADD using *Mentor Graphics Tanner schematic capture* (Tanner 2015). While designing the schematics, we size all the logic gates to template gate according to minimum-size inverter. We then extract netlists from the schematics and simulate the extracted netlists using *Synopsys HSPICE circuit simulator* (HSPICE 2013) with *Predictive Technology Model* (PTM) 32nm model files (PTM 2007). While evaluating the area of RCA and ApproxADD, we neglect area imposed by interconnects and extract layout area of logic gates only from the HSPICE netlists. Further, for assessing the approximate computing design metrics (such as MRED and ER) of N-bit ApproxADD, we perform simulations using (MATLAB 2015) for one million random input patterns drawn from a uniformly distributed sample space between 0 and $2^N - 1$. We carry-out the aforementioned simulation setup for all different versions of ApproxADD (to be discussed in subsequent sections).



(a) Normalized delay, power, area and PDAP

(b) MRED and ER

Fig. 7. Analytical and simulation results of N-bit ApproxADD as a function of N for: (a) Normalized (w.r.t. N-bit RCA) delay, power, area, and PDAP; and (b) MRED (on left y-axis) and ER (on right y-axis) metrics.

4.4 Results and Discussion

The analytical and simulation results of ApproxADD as a function of N for normalized delay, power, area, and PDAP are shown in Figure 7(a) and for MRED and ER are shown in Figure 7(b). It can be seen from Figure 7(a) that the normalized delay of ApproxADD decreases with bit-width at the rate ≈1/N. This implies that ApproxADD provides bit-width-aware constant delay ($O(1)$). ApproxADD also improves dynamic power consumption by 46.31% and area by 28.57% w.r.t. RCA, which exhibits the lowest power and area. Further, as shown in Figure 7(b), MRED of ApproxADD decreases with bit-width, but it saturates for $N \geq 9$. On the other hand, ER of ApproxADD increases rapidly with bit-width (≈1 for $N \geq 13$). It should be noted that the difference between analytical and simulation results (here as well as in rest of the article) is very small. It means our analytical models/results agree closely with simulation results. Further, analytical and simulation results of area are identical because in both cases, we neglect area imposed by interconnects and consider area of logic gates only.

5 IMPROVING ED AND ER METRICS

It can be seen from Figure 7(a) that ApproxADD provides a significant improvement in delay, power, area, and PDAP. However, as shown in Figure 7(b), its: (i) MRED is too high (≈0.4 for $N \geq 9$); and (ii) ER increases rapidly with bit-width (≈1 for $N \geq 13$). Therefore, ApproxADD may not be preferred for applications in which error-resilience is very low. Moreover, according to the approximate computing design concept of *fail small or fail rare* (Chippa et al. 2013), approximate schemes should be constrained either in ED or in ER. In this section, we explain our strategies to improve ED and ER of ApproxADD.

5.1 Concept of Carry-lifetime

We know that in arithmetic operations, impact of errors in higher-order bits is more severe as compared to the errors in lower-order bits. Therefore, to improve ED of ApproxADD, we perform higher-order k-bit operations accurately (see Figure 8). For estimating the number of higher-order k-bits that should be performed accurately, we avail the probability of carry-lifetime. In conventional binary addition process, we define carry-lifetime as the number of bits a carry propagates before annihilation. For example, if a carry generates at the mth bit position and propagates to the pth bit position, then the carry has a lifetime of $p - m$ bits, where $p > m$. For assessing the probability of carry-lifetime, we design N-bit conventional adders for different values of N. We then simulate all the designed adders for one million random input patterns drawn from a uniformly distributed sample space between 0 and $2^N - 1$. Our simulation results for $N = 8,\ 16,\ 32,$

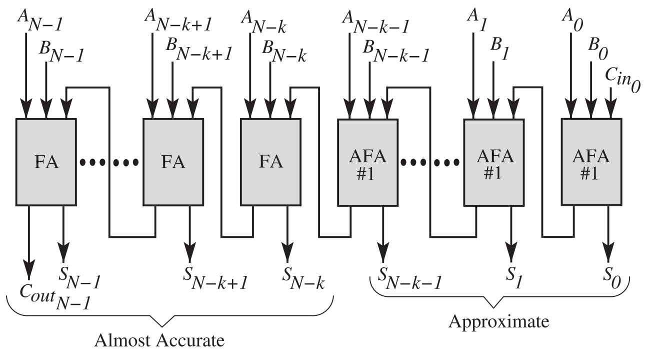


Fig. 8. Architecture of an N-bit ApproxADDv1 designed using k-bit RCA and $(N-k)$-bit ApproxADD. Here, "almost accurate" signifies that ED and ER of higher-order k-bits is very low.

Table 5. Simulation Results of Probability of Carry-lifetime in N-bit Conventional Adders

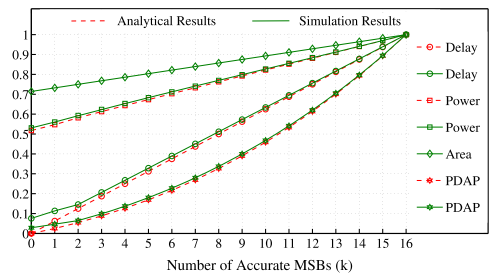
N \ l	1	2	3	4	5	6	7	8	9	10
8	0.5622	0.8121	0.9213	0.9686	0.9881	0.9961	0.9990	1.0000	—	—
16	0.5317	0.7815	0.8985	0.9531	0.9785	0.9902	0.9956	0.9980	0.9991	0.9996
32	0.5161	0.7659	0.8866	0.9450	0.9734	0.9872	0.9938	0.9970	0.9986	0.9993
64	0.5036	0.7544	0.8789	0.9405	0.9708	0.9857	0.9929	0.9965	0.9983	0.9991

and 64 are tabulated in Table 5. It can be seen from Table 5 that the worst-case carry propagation (i.e., carry having lifetime ≈N bits) rarely happens. Therefore, in most of the cases, carry can be determined by considering just few input bits (l) on the right of current bit position (Zhu et al. 2009). For example, in a 16-bit conventional adder, for $l = 1,\ 2$, and 3, the probability of getting correct carry is 0.5317, 0.7815, and 0.8985, respectively. However, if the value of l is increased to 8, 9, and 10, then the probability of getting correct carry is 0.9980, 0.9991, and 0.9996, respectively. Here, it should also be observed from Table 5 that the probability of getting correct carry at ith bit position (when n number of bits on the right of ith bit position are taken into consideration) can be mathematically given by

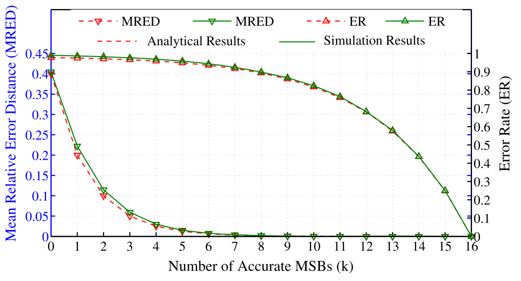
$$P(C_{out_i} = correct) \approx 1 - 0.5^n. \tag{22}$$

Based on the earlier observations, we perform higher-order k-bit operations accurately using conventional adder and lower-order $(N-k)$-bit operations approximately using AFA#1. The higher-order k-bit operations can be performed using any adder architecture. Considering RCA architecture, Figure 8 shows the modified version of ApproxADD, which hereinafter is referred as "ApproxADDv1."

5.1.1 Delay, Power, and Area Analysis. In ApproxADDv1, higher-order k-bit and lower-order $(N-k)$-bit operations are performed simultaneously. Since delay of lower-order $(N-k)$-bits is constant irrespective of bit-width, the worst-case delay of N-bit ApproxADDv1 is same as the k-bit RCA. Consequently, the normalized delay (D_{nm}) of an N-bit ApproxADDv1 w.r.t. RCA can be given by k/N. Further, using Table 3 and Table 4, the normalized dynamic power consumption (P_{nm}) and normalized area (A_{nm}) of an N-bit ApproxADDv1 w.r.t. RCA can be given by



(a) Normalized delay, power, area and PDAP



(b) MRED and ER

Fig. 9. Analytical and simulation results of 16-bit ApproxADDv1 as a function of k: (a) Normalized (w.r.t. 16-bit RCA) delay, power, area, and PDAP; and (b) MRED (on left y-axis) and ER (on right y-axis) metrics.

$$\begin{aligned} P_{nm} &= \frac{k \times P_{nm\_inv_{FA}} + (N-k) \times P_{nm\_inv_{AFA\#1}}}{N \times P_{nm\_inv_{FA}}} \\ &= \frac{k}{N} \times \left(1 - \frac{P_{nm\_inv_{AFA\#1}}}{P_{nm\_inv_{FA}}}\right) + \frac{P_{nm\_inv_{AFA\#1}}}{P_{nm\_inv_{FA}}} \\ &= 0.5275 + \left(0.4725 \times \frac{k}{N}\right), \end{aligned} \tag{23}$$

$$\begin{aligned} A_{nm} &= \frac{k \times A_{FA} + (N-k) \times A_{AFA\#1}}{N \times A_{FA}} \\ &= \frac{k}{N} \times \left(1 - \frac{A_{AFA\#1}}{A_{FA}}\right) + \frac{A_{AFA\#1}}{A_{FA}} \\ &= 0.7143 + \left(0.2857 \times \frac{k}{N}\right). \end{aligned} \tag{24}$$

5.1.2 ED and ER Analysis. In ApproxADDv1, C_{in} of higher-order k-bits is considered as A_{N-k-1}. Based on the discussion presented in Section 3 (Table 2), the probability of C_{in} of higher-order k-bits to be erroneous is 0.25. Further, the probability that this erroneous C_{in} will propagate to MSBs depends on k. For example, as per the simulation results presented in Table 5, in a 16-bit ApproxADDv1, for $k = 7,\ 8,\ 9$, and 10, the probability that erroneous C_{in} will propagate to MSB is $0.0044,\ 0.0020,\ 0.0009$, and 0.0004, respectively. It can be deduced from the above observations that due to cumulative effect, the probability of error in higher-order k-bits is very low (particularly, for $k \geq 8$). Consequently, the error characteristics of N-bit ApproxADDv1 are similar to $(N-k)$-bit ApproxADD, and thus, ED and ER of an N-bit ApproxADDv1 can be given by replacing N with $(N-k)$ in Equation (19) and Equation (21), respectively.

5.1.3 Results and Discussion. The analytical and simulation results of 16-bit ApproxADDv1 as a function of k for normalized delay, power, area, and PDAP are shown in Figure 9(a) and for MRED and ER are shown in Figure 9(b). It can be seen from Figure 7(b) and Figure 9(b) that the approach used in ApproxADDv1 (i.e., to perform higher-order k-bit operations accurately) improves MRED significantly. MRED of an N-bit ApproxADDv1 can empirically be given by $0.4/2^k$. ApproxADDv1 also shows improvement in ER, but the improvement is marginal (particularly, for $k \leq 8$). Here, it should be noted that MRED of ApproxADDv1 decreases with k, whereas its normalized delay, power, area, and PDAP increase with k. It means ApproxADDv1 provides improvement in MRED at the cost of delay, power, area, and PDAP. However, the rate at which MRED decreases is

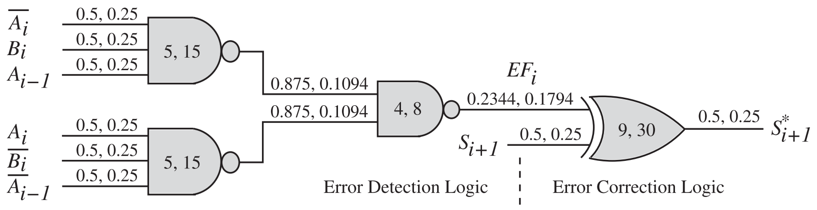


Fig. 10. Gate-level logic implementation of the proposed EDC logic. The numbers indicated on the logic gates and on the input and output nodes of the logic gates have been described earlier in Figure 3.

Table 6. Design Parameters of the Logic Gates Used in EDC Logic (Figure 10)

Logic Gates	α	g	h	p_d	p_p	a	d	p_{nm_inv}
NAND	0.1094	5/3	4/5	3	3	15	4.33	1.89
NAND	0.1794	4/3	9/4	2	2	8	5	3.59
XOR	0.25	3	1/3#	4	5	30	5	6

Here, # signifies that while computing the electrical effort (h) of output logic gates, we assume that S^* drives a unit standard CMOS inverter.

higher as compared to the rate at which delay, power, area, and PDAP increases. This shows that ApproxADDv1 improves MRED in a graceful manner.

5.2 Error Detection and Correction Logic

Although ApproxADDv1 provides a significant improvement in ED, it may still be not preferred for some of the error-resilient applications (particularly, for applications which demand low ER). To improve ER of ApproxADDv1, we exploit the concept of error detection and correction in which an EDC logic detects errors and accordingly corrects them. An N-bit ApproxADDv1 can be considered as k-bit RCA followed by $(N-k)$-bit ApproxADD (see Figure 8). As discussed earlier, the probability of error in higher-order k-bits is very low. Further, in lower-order $(N-k)$-bits, an erroneous carry generates at ith bit position only when $\overline{A}_i.B_i.A_{i-1} = 1$ or $A_i.\overline{B}_i.\overline{A}_{i-1} = 1$, where $i \leq N-k$. The occurrence of such input combinations can be divulged using the error detection logic, as shown in Figure 10. Here, true value of *Error Flag* (EF_i) indicates that an erroneous carry is generated at ith bit position. We know that in arithmetic operations, erroneous carry generated at ith bit position affects the output at $(i+1)$th bit position. Therefore, output at $(i+1)$th bit position can be corrected (S_{i+1} is complemented when $EF_i = 1$) using the error correction logic, as shown in Figure 10. The design parameters, such as α, g, h, p_d, p_p, a, d, and p_{nm_inv} of all the logic gates used in EDC logic are tabulated in Table 6. Using Table 6, the EDC logic has delay $D = 4.33 + 5 + 5 = 14.33$, normalized dynamic power consumption $P_{nm\_inv} = 1.89 + 1.89 + 3.59 + 6 = 13.37$, and area $A = 15 + 15 + 8 + 30 = 68$ units.

Now we integrate ApproxADDv1 and EDC logic, which hereinafter is referred as "ApproxADDv2." As shown in Figure 11, ApproxADDv2 has two stages. In first stage, sum bits are computed using k-bit RCA and $(N-k)$-bit ApproxADD. Since error conditions can be detected simultaneously with sum bits, an $(N-k)$-bit error detection logic is also implemented in first stage to actuate EFs. In second stage, depending on EFs, errors are corrected using an $(N-k)$-bit error correction logic.

5.2.1 Delay, Power, and Area Analysis. In ApproxADDv2, higher-order k-bit and lower-order $(N-k)$-bit operations are performed simultaneously. Since delay of lower-order $(N-k)$-bits is

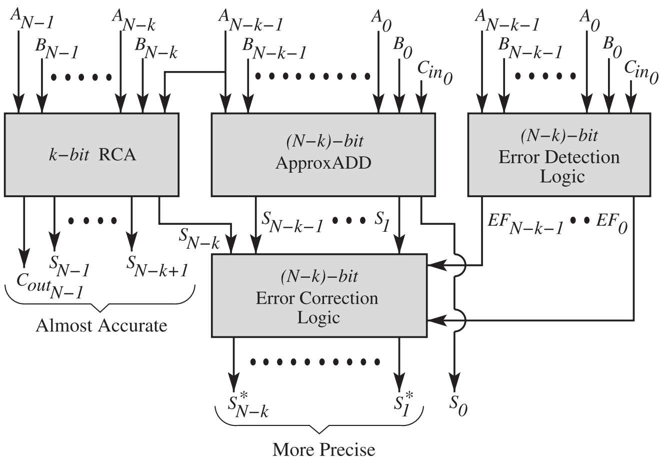


Fig. 11. Architecture of an N-bit ApproxADD$v2$ designed using k-bit RCA, $(N-k)$-bit ApproxADD, and $(N-k)$-bit EDC logic. Here, "more precise" signifies that the probability of error in $S^*_{N-k:1}$ is less as compared to the probability of error in $S_{N-k:1}$.

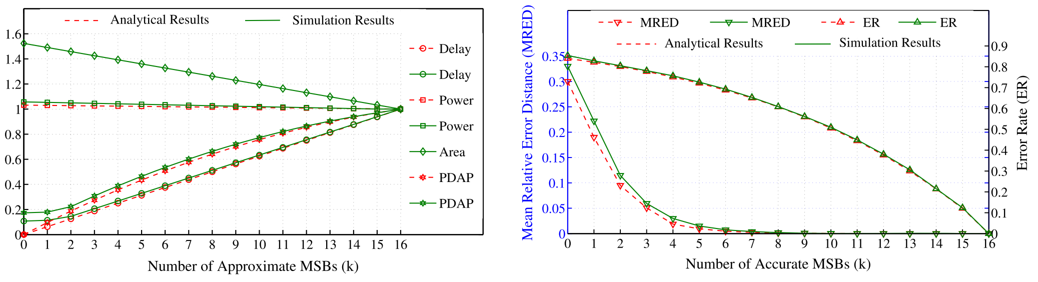
constant irrespective of bit-width, the worst-case delay of N-bit ApproxADD$v2$ is same as the k-bit RCA. Consequently, the normalized delay (D_{nm}) of an N-bit ApproxADD$v2$ w.r.t. RCA can be given by k/N. Further, using Table 3 and Table 6, the normalized dynamic power consumption (P_{nm}) and normalized area (A_{nm}) of an N-bit ApproxADD$v2$ w.r.t. RCA can be given by

$$\begin{aligned} P_{nm} &= \frac{k \times P_{nm\_inv_{FA}} + (N-k) \times (P_{nm\_inv_{AFA\#1}} + P_{nm\_inv_{EDC}})}{N \times P_{nm\_inv_{FA}}} \\ &= 1.0313 - \left(0.0313 \times \frac{k}{N}\right), \end{aligned} \tag{25}$$

$$\begin{aligned} A_{nm} &= \frac{k \times A_{FA} + (N-k) \times (A_{AFA\#1} + A_{EDC})}{N \times A_{FA}} \\ &= 1.5238 - \left(0.5238 \times \frac{k}{N}\right), \end{aligned} \tag{26}$$

where $P_{nm\_inv_{EDC}}$ is the normalized dynamic power consumption of EDC logic w.r.t. a unit standard CMOS inverter and A_{EDC} is the area of EDC logic. Equation (25) and Equation (26) show that ApproxADD$v2$ consumes more power and area as compared to RCA. But note that power and area overheads decrease with k. Further, the rate at which power and area overheads decrease depends on bit-width.

5.2.2 ED and ER Analysis. As shown in Figure 11, an N-bit ApproxADD$v2$ can be considered as k-bit RCA followed by $(N-k)$-bit ApproxADD and $(N-k)$-bit EDC logic. We know that in ApproxADD, two consecutive erroneous carries generated from ith bit position do not introduce error at the $(i+2)$th bit position. However, the EDC logic complements output at the $(i+2)$th bit position, regardless of the output at the $(i+2)$th bit position is correct (refer Figure 11 in Dutt et al. (2016b)). This implies that the EDC logic corrects error when input combinations $\overline{A}_i.B_i.A_{i-1}$ and $A_i.\overline{B}_i.\overline{A}_{i-1}$ occur separately, but introduces errors when input combinations $\overline{A}_i.B_i.A_{i-1}$ and $A_i.\overline{B}_i.\overline{A}_{i-1}$ occur consecutively. Therefore, ED of an N-bit ApproxADD$v2$ can be given by the weighted sum of input combinations that generate consecutive erroneous carries as



(a) Normalized delay, power, area and PDAP (b) MRED and ER

Fig. 12. Analytical and simulation results of 16-bit ApproxADDv2 as a function of k: (a) Normalized (w.r.t. 16-bit RCA) delay, power, area, and PDAP; and (b) MRED (on left y-axis) and ER (on right y-axis) metrics.

$$ED \approx \sum_{i=0}^{N-k-1} 2^{i+1} \times \left\{(\overline{A}_i.B_i.A_{i-1}) \;.\; (A_{i-1}.\overline{B}_{i-1}.\overline{A}_{i-2})\right\} + \sum_{i=0}^{N-k-1} 2^{i+1} \times \left\{(A_i.\overline{B}_i.\overline{A}_{i-1}) \;.\; (\overline{A}_{i-1}.B_{i-1}.A_{i-2})\right\}. \tag{27}$$

Further, to model ER of ApproxADDv2, we use an empirical approach, that is, we model ER using simulation results. From our simulation results, we observe that ER of ApproxADDv2 forms a GP with scalar factor ≈1/9 and common ratio ≈8/9. Accordingly, ER of an N-bit ApproxADDv2 can be given by

$$ER = \sum_{i=1}^{N} \frac{1}{9} \times \left(\frac{8}{9}\right)^{i-1} = \sum_{i=1}^{N} \frac{8^{i-1}}{9^i}. \tag{28}$$

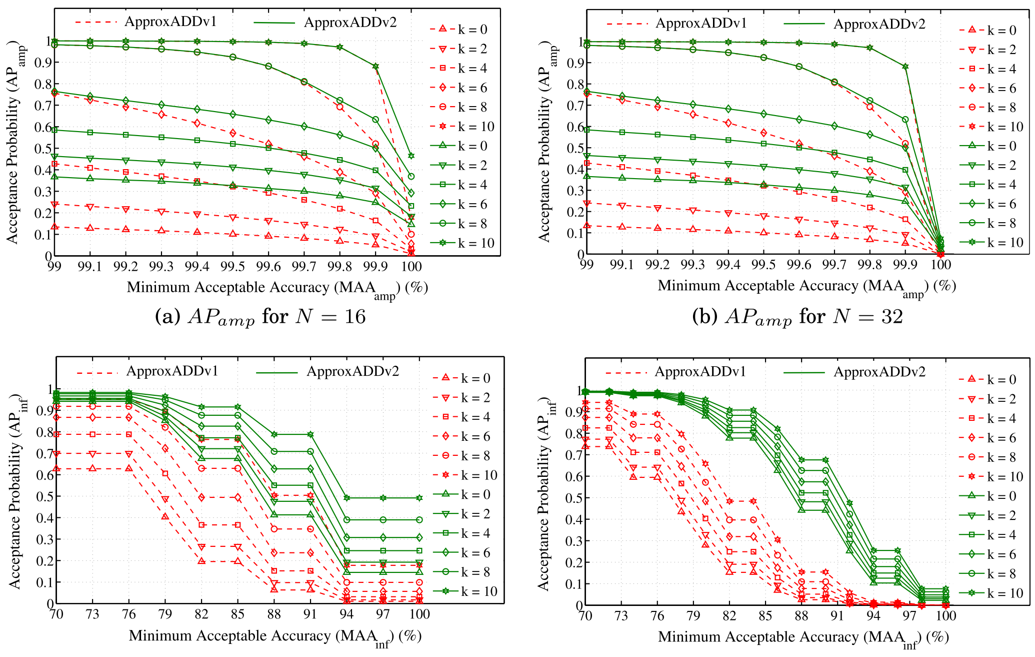
5.2.3 *Results and Discussion.* The analytical and simulation results of 16-bit ApproxADDv2 as a function of k for normalized delay, power, area, and PDAP are shown in Figure 12(a) and for MRED and ER are shown in Figure 12(b). It can be seen from Figure 9(b) and Figure 12(b) that ApproxADDv2 improves ER significantly. ApproxADDv2 provides this improvement at the cost of power and area overheads (see Figure 9(a) and Figure 12(a)). Although ApproxADDv2 imposes power and area overheads, it shows a significant improvement in overall performance, which is generally measured in terms of PDAP. Further, in case of parallel-prefix adders (discussed in Section 6.1), the proposed approach also provides improvement in power as well as area.

6 PERFORMANCE ANALYSIS

For evaluating the effectiveness of approximate schemes, researchers/designers introduce several approximate computing design metrics (Liang et al. 2013; C. Liu and Lombardi 2015). Three of the approximate computing design metrics (ED, MRED, and ER) and simulation results of the proposed approach w.r.t. these metrics have been discussed in previous sections. Besides these metrics, in error-resilient *Digital Signal Processing* (DSP) applications, *Accuracy of Amplitude* (ACC$_{amp}$) (Kahng and Kang 2012) is used to evaluate EDs for the amplitude data as

$$ACC_{amp}(\%) = \left(1 - \frac{ED}{R_c}\right) \times 100. \tag{29}$$

On the other hand, in error-resilient communication applications that mainly handle information data, the number of incorrect bits, that is, Hamming distance is more meaningful. For such



(a) AP_{amp} for $N = 16$ (b) AP_{amp} for $N = 32$

(c) AP_{inf} for $N = 16$ (d) AP_{inf} for $N = 32$

Fig. 13. Acceptance probabilities (AP_{amp} and AP_{inf}) of 16-bit and 32-bit ApproxADD$v1$ and ApproxADD$v2$ as a function of k ($k = 0,\ 2,\ 4,\ 6,\ 8,$ and 10): (a) AP_{amp} of 16-bit ApproxADD$v1$ and ApproxADD$v2$; (b) AP_{amp} of 32-bit ApproxADD$v1$ and ApproxADD$v2$; (c) AP_{inf} of 16-bit ApproxADD$v1$ and ApproxADD$v2$; and (d) AP_{inf} of 32-bit ApproxADD$v1$ and ApproxADD$v2$.

applications, *Accuracy of Information* (ACC$_{inf}$) (Kahng and Kang 2012) is used to evaluate EDs for the information data as

$$ACC_{inf}(\%) = \left(1 - \frac{B_e}{N}\right) \times 100, \tag{30}$$

where B_e is the number of incorrect bits. Further, errors are allowed to exist at output in error-resilient applications provided that the accuracy of output is higher than (or at least equal to) the acceptance threshold. Different applications have a different level of acceptance threshold depending on their inherent error-resilience (Chippa et al. 2013). Researchers measure acceptance threshold in different ways: (i) (Shin and Gupta 2008) measure acceptance threshold in terms of *Distance Rate* (DR), that is, product of ED and ER; and (ii) (Zhu et al. 2009) measure acceptance threshold in terms of *Minimum Acceptable Accuracy* (MAA). If MAA represents acceptance threshold, then the criterion of acceptability is given by the *Acceptance Probability* (AP) as

$$AP_{amp} = P(ACC_{amp} \geq MAA_{amp}), \tag{31}$$
$$AP_{inf} = P(ACC_{inf} \geq MAA_{inf}), \tag{32}$$

where AP$_{amp}$ and MAA$_{amp}$ are the AP and MAA for amplitude data, and AP$_{inf}$ and MAA$_{inf}$ are the AP and MAA for information data, respectively. To inspect effectiveness of the proposed approach, we simulate ApproxADD$v1$ and ApproxADD$v2$ over the aforementioned approximate computing design metrics. Our simulation results for AP$_{amp}$ and AP$_{inf}$ of 16-bit and 32-bit ApproxADD$v1$ and ApproxADD$v2$ as a function of k are shown in Figure 13. Further, for some particular values of DR (0.001%, 0.01%, 0.1%, 1%, and 10%), Figure 14 shows the relationship between N and k for ApproxADDv2. From Figure 13 and Figure 14, we observe that:

(1) Figure 13(a) and Figure 13(b) show that for a fixed value of k, both ApproxADDv1 and ApproxADDv2 provide similar AP$_{amp}$ for $N = 16$ and 32 (except for MAA$_{amp}$ = 100%). This implies that both in ApproxADDv1 and ApproxADDv2, for a fixed value of k, AP$_{amp}$ does not depend on bit-width. In other words, for a particular value of AP$_{amp}$, k is independent of bit-width. Therefore, in case of AP$_{amp}$, ApproxADDv1 and ApproxADDv2 provide bit-width-aware constant delay.

(2) Figure 13(a) and Figure 13(b) show that for higher values of k ($k \geq 10$), ApproxADDv1 and ApproxADDv2 provide similar AP$_{amp}$ (except for MAA$_{amp}$ = 100%). Therefore, in error-resilient DSP applications that demand high AP$_{amp}$, ApproxADDv1 is preferred over ApproxADDv2 because both ApproxADDv1 and ApproxADDv2 provide similar AP$_{amp}$, but ApproxADDv1 consumes less power and area as compared to ApproxADDv2 (see Figure 9(a) and Figure 12(a)). On the other hand, Figure 13(c) and Figure 13(d) show that for a fixed value of k, ApproxADDv2 provides higher AP$_{inf}$ than ApproxADDv1. Therefore, in error-resilient communication applications that demand high AP$_{inf}$, ApproxADDv2 is preferred over ApproxADDv1.

(3) Figure 14 shows that for higher values of DR (DR $\geq$ 1%), k is almost independent of N, whereas for smaller values of DR (DR $\leq$ 0.01%), k varies linearly with N. This implies that for smaller values of DR, delay of ApproxADDv2 varies linearly with bit-width. However, if constraints on DR are relaxed, then ApproxADDv2 provides bit-width-aware constant delay. Our simulation results show that ApproxADDv1 also exhibits similar behavior, but ApproxADDv1 provides bit-width-aware constant delay for higher values of DR as compared to ApproxADDv2.

6.1 Effect of Higher-Order k-bit Adder Architecture

The final version of the proposed approximate adder (i.e., ApproxADDv2) is shown in Figure 11, where we perform higher-order k-bit operations using RCA and lower-order $(N - k)$-bit operations using ApproxADD and EDC logic. For the ease of illustration, we use RCA architecture for higher-order k-bit operations. However, the higher-order k-bit operations can be performed using any adder architecture. To evaluate how effectiveness of the proposed approach varies with higher-order k-bit adder architectures, we perform higher-order k-bit operations using *Brent-Kung* (BK) (Brent and Kung 1982), *Kogge-Stone* (KS) (Kogge and Stone 1973), and *Sklansky* (Sk) (Sklansky 1960) architectures. We choose BK, KS, and Sk because these are the fundamental and most widely used parallel-prefix adders. We call the resulting approximate adders as ApproxBK, ApproxKS, and ApproxSk, respectively.

Since approximation is introduced in lower-order $(N - k)$-bits only, the adder architecture used for higher-order k-bits makes no difference to the approximate computing design metrics. Therefore, the analytical as well as simulation results shown in Figure 12(b) and Figure 13 are valid for ApproxBK, ApproxKS, and ApproxSk also. On the other hand, different adder architectures have different delay, power, and area. Therefore, the adder architecture used for higher-order k-bits shows a significant difference in delay, power, and area. The simulation results for normalized delay, power, and area of 16-bit ApproxBK, ApproxKS, and ApproxSk w.r.t. BK, KS, and Sk adders, respectively, as a function of k are shown in Figure 15. It can be seen from Figure 12(a) and Figure 15 that the higher-order k-bit adder architecture shows contradictory behavior. In Figure 12(a), normalized power and area are higher than 1 and decrease with k, whereas in Figure 15, normalized power and area are lower than 1 and increase with k. This is because the proposed approximate

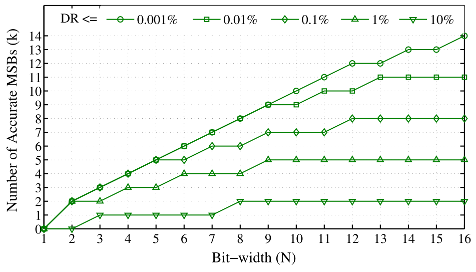


Fig. 14. For some particular values of DR (0.001%, 0.01%, 0.1%, 1%, and 10%), relationship between N and k for N-bit ApproxADDv2, where $N \leq 16$.

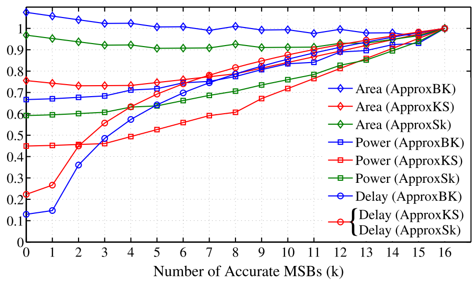


Fig. 15. Normalized delay, power, and area of 16-bit ApproxBK, ApproxKS, and ApproxSk w.r.t. BK, KS, and Sk adders, respectively, as a function of k.

Table 7. For $k = 8$ and 10, the Percentage Change (Δ) in D, P, A, and $PDAP$ of 16-bit ApproxADDv2, ApproxBK, ApproxKS, and ApproxSk w.r.t. 16-bit RCA, BK, KS, and Sk Adders, Respectively

Approximate Adders	$k = 8$				$k = 10$			
	ΔD	ΔP	ΔA	$\Delta PDAP$	ΔD	ΔP	ΔA	$\Delta PDAP$
ApproxADDv2	-48.840	3.0687	26.190	-39.880	-36.630	2.3112	19.642	-27.985
ApproxBK	-21.319	-22.537	0.9973	-38.444	-14.454	-16.536	-0.6299	-29.050
ApproxKS	-18.340	-39.284	-21.254	-60.957	-12.441	-28.152	-15.940	-47.119
ApproxSk	-18.340	-29.347	-7.3758	-46.560	-12.441	-24.018	-8.8416	-39.353

cell (AFA#1 + EDC) has overall power and area more than FA cell but lower than BK, KS, and Sk cells. Further, the rate at which normalized delay, power, and area changes also depends on the higher-order k-bit adder architecture. Here, it should also be noted that if we perform higher-order k-bit operations using RCA architecture, then the proposed approach imposes power and area overheads. However, if we perform higher-order k-bit operations using state-of-the-art adder architectures (such as BK, KS, and Sk), then the proposed approach provides a significant improvement in power as well as area. Further, as discussed in Section 5.1.2, ED and ER of higher-order k-bits is very low for $k \geq 8$. Therefore, for $k = 8$ and 10, the percentage change in delay (ΔD), power (ΔP), area (ΔA), and PDAP ($\Delta PDAP$) of 16-bit ApproxADDv2, ApproxBK, ApproxKS, and ApproxSk w.r.t. RCA, BK, KS, and Sk adders, respectively, are tabulated in Table 7. It can be seen from Table 7 that the proposed approach provides the highest improvement in delay for RCA architecture and in power, area, and PDAP for KS architecture.

6.2 Comparison with Existing Approximate Adders

Based on the motivation discussed in Section 1, several approximate adders have been proposed in literature. For evaluating efficiency of the proposed approach, we compare ApproxADDv1 and ApproxADDv2 with existing approximate adders. For comparison purposes, we choose *Error-tolerant Adders* (ETAs) (Zhu et al. 2010, 2009), *Approximate Carry Skip Adder* (ACSA) (Kim et al. 2013), *Yang's Adder* (YA) (Yang et al. 2016), *Lu's Adder* (LA) (Lu 2004), *Accuracy Configurable Adder* (ACA) (Kahng and Kang 2012), *Simplified Full Adder* (SFA) (Shin and Gupta 2008), *Low-part-OR Adder* (LOA) (Mahdiani et al. 2010), and *Soares's Adder* (SA) (Soares et al. 2015). To have a fair comparison: (i) We design all these approximate adders using RCA architecture, and (ii) while designing, we keep the maximum length of carry propagation same in all these approximate adders.

Table 8. Comparison of the Proposed Approach with Existing Approximate Adders

Design Metrics	RCA	Equal Segment-Based Approximate Adders					AFA-Based Approximate Adders					
		ETA-II	ACSA	YA	LA	ACA	ETA-I	SFA	LOA	SA	Approx ADD$v1$	Approx ADD$v2$
Delay (ps)	895.06	335.64	344.63#	349.04#	335.64	335.64	335.64	335.64	335.64	335.64	335.64	335.64
Power (μW)	347.82	378.12	389.35	382.52	984.49	592.68	272.17	213.84	187.81	173.43	224.53	349.78
Area (μm^2)	236.37	327.91	351.89	336.10	729.69	408.82	236.28	183.61	112.47	101.79	194.16	313.76
PDAP	7.4E+7	4.2E+7	4.7E+7	4.4E+7	2.4E+8	8.1E+7	2.2E+7	1.3E+7	7.1E+6	5.9E+6	1.5E+7	3.7E+7
MRED	0.0000	2.7E-4	2.4E-4	6.1E-5	1.5E-5	1.7E-5	1.4E-4	3.3E-4	1.2E-4	1.6E-4	9.5E-5	2.5E-5
ER	0.0000	0.8747	0.8496	0.7904	0.0025	0.0292	0.9991	0.9733	0.9992	0.9999	0.9967	0.8745
Acc_{amp} (avg.)	1.0000	0.9973	0.9981	0.9992	0.9999	0.9998	0.9986	0.9993	0.9990	0.9982	0.9991	0.9997
Acc_{inf} (avg.)	1.0000	0.9064	0.9271	0.9407	0.9999	0.9982	0.6295	0.7303	0.6049	0.5817	0.8362	0.9221

# Although we keep the maximum length of carry propagation same while designing all these approximate adders, the delay of ACSA and YA is higher as compared to other approximate adders. This is due to additional delay imposed by multiplexers and flip-flops used in ACSA and YA.

Table 8 shows our simulation results for conventional design metrics and approximate computing design metrics of RCA (as a reference) and all the aforementioned approximate adders along with ApproxADD$v1$ and ApproxADD$v2$ for $N = 32$ and $k = 12$. For the ease of comparison, we classify these approximate adders (based on their design approach) into two categories: (i) Equal segment-based approximate adders; and (ii) AFA-based approximate adders. Note that as compared to AFA-based approximate adders, the design approach of ETA-I is different. But, since error characteristics of ETA-I are similar to AFA-based approximate adders, we keep it in the same category. From Table 8, we observe that

(1) LA and ACA provide the smallest MRED and ER, and the highest ACC_{amp} and ACC_{inf}. Therefore, these are suitable for applications in which error resilience is very low. However, LA and ACA impose sever power and area overheads (due to multitude of sub-adders required in their implementation) that prevent their use in low-power and area-constrained applications.
(2) ETA-I, SFA, LOA, SA, and ApproxADD$v1$ provide higher ER and smaller ACC_{inf}. Therefore, these are suitable for applications in which error resilience is very high. As compared to SFA, LOA, and SA, ApproxADD$v1$ consumes more power and area, but it provides higher ACC_{inf} than SFA, LOA, and SA. Consequently, in error-resilient communication applications (such as speech, audio, and video communications over wired/wireless networks (Prasad and Mihovska 2009)), ApproxADD$v1$ is preferred over SFA, LOA, and SA. Moreover, ApproxADD$v1$ provides better MRED, ER, and ACC_{amp} than LOA and SA. Further, as compared to ETA-I, ApproxADD$v1$ provides smaller power and area, smaller MRED and ER, and higher ACC_{amp} and ACC_{inf}. Therefore, both in error-resilient DSP applications and communication applications, ApproxADD$v1$ is preferred over ETA-I.
(3) ETA-II, ACSA, YA, and ApproxADD$v2$ are suitable for applications in which error resilience is moderate. As compared to ApproxADD$v2$, ACSA and YA provide better ER and ACC_{inf}, but ACSA and YA exhibit higher delay, power, and area. Therefore, in high-performance low-power applications, ApproxADD$v2$ is preferred over ACSA and YA. Further, ETA-II and ApproxADD$v2$ have similar ER, but ApproxADD$v2$ provides smaller power and area, smaller MRED, and higher ACC_{amp} and ACC_{inf} than ETA-II. Therefore, both in error-resilient DSP applications and communication applications, ApproxADD$v2$ is preferred over ETA-II.

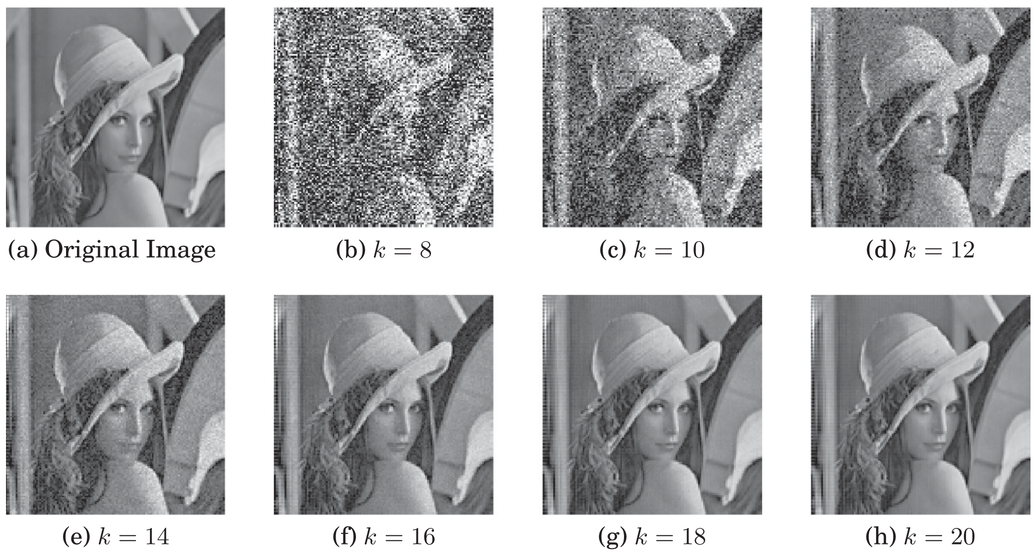


Fig. 16. Image processing results: (a) Original image; and images after performing DCT and IDCT using 24-bit ApproxADDv2 for: (b) $k = 8$; (c) $k = 10$; (d) $k = 12$; (e) $k = 14$; (f) $k = 16$; (g) $k = 18$; and (h) $k = 20$.

7 MULTIMEDIA APPLICATIONS: A CASE STUDY

We use many electronic systems equipped with multimedia applications that are integral part of our lives. One of the critical challenges is to operate such electronic systems for real-time processing while meeting the design specifications. Since CMOS technology scaling is becoming less and less effective at improving system capability, researchers/designers have started exploring alternate avenues to address such challenges. Approximate computing is considered the most suitable alternate avenue for designing electronic systems for multimedia applications because the limited perception of human senses removes the strict restrictions on accuracy and allows the output of multimedia applications to be approximate. Moreover, in most of the multimedia applications, commonly used image processing algorithms, such as JPEG have inherent error resilience. In such algorithms, DCT and IDCT are integral components that involve a large number of addition and multiplication operations. Since multiplication is generally computed using shift and add operations, adders can be treated as the basic building blocks of DCT and IDCT modules. To inspect effectiveness of the proposed approach in real-life applications, we replace the conventional addition operations in DCT and IDCT modules with ApproxADDv2. For evaluating effectiveness of image processing approaches, several standard test images have been introduced in the literature (USC-SIPI 1977). Among all existing test images, Lena picture is the most widely used standard test image. We also use the same Lena picture in our analysis. Further, there are two principle methods used for evaluating image quality: (i) subjective; and (ii) objective. To have a fair evaluation of image quality, we evaluate output images both subjectively as well as objectively.

In the subjective method, image quality is evaluated by humans. For the sake of subjective evaluation, Figure 16 shows the original Lena image and output images after performing DCT and IDCT using 24-bit ApproxADDv2 for different values of k. It can be seen from Figure 16 that for $k \geq 16$, visual quality loss to the output images is negligible. For $12 \leq k \leq 14$, output images show some blockiness that may be tolerated by human eyes, and thus, may be acceptable for some of the error-resilient applications. However, with further decrease in k (particularly, for $k < 10$), there is a severe degradation in image quality that is (generally) not acceptable.

Table 9. Error Characteristics and IQA Metrics of the Output Images after Performing DCT and IDCT Using 24-bit ApproxADDv2 for Different Values of k

Quality Metrics			$k = 8$	$k = 10$	$k = 12$	$k = 14$	$k = 16$	$k = 18$	$k = 20$
Approx-ADDv2	Error	μ	-1.41E-14	-1.37E-14	-1.34E-14	-1.32E-14	-1.30E-14	-1.29E-14	-1.28E-14
		σ	153.3015	121.8938	98.2511	86.1178	78.1383	73.5977	69.5073
	MSE		2.35E+4	1.12E+4	7.69E+3	4.30E+3	2.09E+3	9.13E+2	4.81E+2
	PSNR (dB)		4.42031	7.62429	9.27493	11.7951	14.9168	18.5239	21.3007
	SSIM		0.06187	0.21220	0.42724	0.61751	0.76165	0.85263	0.89768
ETA-I	MSE		5.01E+4	3.18E+4	1.99E+4	9.49E+3	3.92E+3	1.38E+3	6.25E+2
	PSNR (dB)		1.12654	3.10157	5.12845	8.35362	12.1927	16.7026	20.1664
	SSIM		0.01253	0.04606	0.10136	0.32812	0.58895	0.80081	0.88745
SFA	MSE		3.81E+4	1.83E+4	1.20E+4	6.30E+3	2.79E+3	1.07E+3	5.22E+2
	PSNR (dB)		2.31735	5.49011	7.31658	10.1319	13.6710	17.8206	20.9464
	SSIM		0.03368	0.12530	0.20078	0.51543	0.70428	0.82775	0.89049
LOA	MSE		5.55E+4	4.18E+4	2.48E+4	1.25E+4	4.93E+3	1.67E+3	6.66E+2
	PSNR (dB)		0.68401	1.91422	4.17132	7.14613	11.1946	15.8931	19.8960
	SSIM		0.00776	0.02188	0.05834	0.16709	0.54232	0.78979	0.88464
SA	MSE		6.07E+4	5.25E+4	3.21E+4	1.42E+4	5.45E+3	1.84E+3	7.11E+2
	PSNR (dB)		0.29431	0.92786	3.05849	6.59293	10.7662	15.4752	19.6092
	SSIM		0.00130	0.00877	0.03892	0.12190	0.53621	0.75788	0.88161

Table also summarizes IQA metrics of the output images after performing DCT and IDCT using 24-bit existing AFA based approximate adders for different values of k.

The subjective method is the only "correct" method of evaluating image quality (Kudelka 2012). However, this method is very expensive and time-consuming. Besides this, evaluation of image quality using the subjective method is not feasible for real-time applications. In such a scenario, domain experts evaluate image quality using the objective method. In objective method, image quality is evaluated by automated computer programs using *Image Quality Assessment* (IQA) metrics. Based on the requirements of applications, several IQA metrics have been proposed in the literature (Wang and Bovik 2006). Two fundamental and most commonly used IQA metrics are *Mean Squared Error* (MSE) and *Peak Signal-to-Noise Ratio* (PSNR), where MSE is a measure of the cumulative squared error and PSNR is a measure of the peak error between the original and processed images. Besides MSE and PSNR, another commonly used state-of-the-art IQA metric is *Structural Similarity Index Metric* (SSIM), which is a measure of the similarity between the original and processed images.

We evaluate output images (shown in Figure 16) for all the aforementioned IQA metrics. Our simulation results are tabulated in Table 9. It can be seen from Table 9 that for $k \leq 10$, values of IQA metrics are not acceptable (in particular the value of SSIM). For $12 \leq k \leq 14$, values of IQA metrics may be acceptable for some of the error-resilient applications. However, for $k \geq 16$, the 24-bit ApproxADDv2 shows comparable (w.r.t. accurate adder) values of IQA metrics. Further, Table 9 also summarizes error characteristics (mean (μ) and standard deviation (σ)) of the output images shown in Figure 16. Here, it should be noted that mean of the error is very close to zero. This implies that the proposed approach overestimates and underestimates the accurate results likely, that is, error is equally distributed around zero. Therefore, the proposed approach is very suitable for computing the operations in which adder is first used for multiple times and then the output (of adder) is used for further processing. One example of such operations is matrix multiplication. If ApproxADDv2 is used in such operations, then the overall error is expected to be negligible ($\approx 10^{-14}$).

For the sake of comparison at application level, we perform DCT and IDCT using existing approximate adders. For comparison purposes, we include AFA-based approximate adders only, that is, ETA-I, SFA, LOA, and SA. Our simulation results of IQA metrics after performing DCT and

Table 10. Error Characteristics and IQA Metrics of the Output Images (Cameraman, Mandrill, and Girl) after Performing DCT and IDCT Using 24-Bit ApproxADD$v2$ for Different Values of k

Quality Metrics			$k = 8$	$k = 10$	$k = 12$	$k = 14$	$k = 16$	$k = 18$	$k = 20$
Camera-man	Error	μ	-1.67E-14	-1.60E-14	-1.56E-14	-1.51E-14	-1.48E-14	-1.46E-14	-1.45E-14
		σ	168.0842	140.8599	114.2347	99.3723	84.6212	78.4631	72.2417
	MSE		2.51E+4	1.20E+4	8.42E+3	4.65E+3	2.29E+3	9.71E+2	5.26E+2
	PSNR (dB)		4.11937	7.33169	8.87510	11.4550	14.5248	18.2573	20.9134
	SSIM		0.05767	0.20456	0.40822	0.59874	0.74183	0.84235	0.88213
Mandrill	Error	μ	-1.18E-14	-1.14E-14	-1.11E-14	-1.08E-14	-1.06E-14	-1.05E-14	-1.04E-14
		σ	131.9697	102.8413	88.1139	78.4913	69.4467	66.4362	64.4001
	MSE		1.97E+4	9.69E+3	6.79E+3	3.52E+3	1.71E+3	7.36E+2	3.82E+2
	PSNR (dB)		5.16941	8.26603	9.81173	12.6629	15.7841	19.4589	22.3059
	SSIM		0.07496	0.23071	0.45167	0.66942	0.79542	0.88678	0.93004
Girl	Error	μ	-9.83E-15	-9.79E-15	-9.76E-15	-9.74E-15	-9.72E-15	-9.71E-15	-9.71E-15
		σ	119.7677	94.4769	80.6031	71.9206	63.9084	61.5285	62.2895
	MSE		1.36E+4	7.04E+3	4.61E+3	2.56E+3	1.22E+3	4.94E+2	2.60E+2
	PSNR (dB)		6.78530	9.65301	11.4845	14.0346	17.2634	21.1869	23.9704
	SSIM		0.09471	0.28663	0.52942	0.72547	0.86614	0.91204	0.95101

IDCT using 24-bit aforementioned approximate adders for different values of k are tabulated in Table 9. It can be seen from Table 9 that ApproxADD$v2$ provides better IQA metrics (smaller MSE, and higher PSNR and SSIM) as compared to ETA-I, SFA, LOA, and SA. Note that the improvement in IQA metrics (due to proposed approach w.r.t. ETA-I, SFA, LOA, and SA) depends significantly on k. As k increases, less number of bits are left for approximation, and thus, improvement in IQA metrics (due to proposed approach w.r.t. ETA-I, SFA, LOA, and SA) decreases. However, for $k \leq 14$, ApproxADD$v2$ provides a significant improvement in IQA metrics (in particular, w.r.t. ETA-I, LOA, and SA).

Further, effectiveness of approximate adders depends on input data types. Based on brightness, sharpness, contrast, and so on, different images represent different data types. Therefore, to have a fair evaluation of the proposed approach, we perform DCT and IDCT using 24-bit ApproxADD$v2$ for different images. Besides Lena, other commonly used standard test images are Cameraman, Mandrill, and Girl (USC-SIPI 1977). Our simulation results of IQA metrics after performing DCT and IDCT using 24-bit ApproxADD$v2$ for the aforementioned images are tabulated in Table 10. It can be seen from Table 9 and Table 10 that for different images, ApproxADD$v2$ provides different IQA metrics. ApproxADD$v2$ provides better IQA metrics for Girl and Mandrill images as compared to Lena and Cameraman images.

8 CONCLUSION

In this article, we proposed four AFAs by modifying conventional FA with design objective that C_{out} is independent of C_{in} subjected to minimal error rate. We then exploit one of the proposed AFAs to construct an N-bit ApproxADD. An emergent property of ApproxADD is that carries do not propagate in it, and therefore, it provides bit-width-aware constant delay ($O(1)$). ApproxADD also improves dynamic power consumption by 46.31% and area by 28.57% w.r.t. RCA, which exhibits the lowest power and area. To improve ED and ER of ApproxADD, we avail the concept of carry-lifetime and EDC logic, respectively. In this way, we introduce two more (improved) versions of ApproxADD – ApproxADD$v1$ and ApproxADD$v2$. Our simulation results showed that if constraints on approximate computing design metrics are slightly relaxed, then ApproxADD$v1$ and ApproxADD$v2$ also provide bit-width-aware constant delay. For the ease of illustration, we discussed the proposed approach w.r.t. RCA, however, we demonstrated that the proposed approach can be customized for other adder architectures. For some of the adder architectures (e.g., RCA), the final version of the proposed approximate adder imposes minor power and area overheads, but it shows significant improvement in overall performance, which is generally measured in terms of PDAP. Further, for state-of-the-art adder architectures (e.g., BK, KS, and Sk), the final version of the proposed approximate adder shows significant improvement in power as well as area. As compared to other AFA-based approximate adders (ETA-I, SFA, LOA, and SA), ApproxADDs provide smaller MRED and ER and higher ACC_{amp} and ACC_{inf}. Therefore, both in error-resilient DSP applications and communication applications, ApproxADDs are preferred over existing AFA-based approximate adders. Further, to inspect effectiveness of the proposed approach in real-life applications, we demonstrated image compression and decompression by replacing the conventional addition operations in DCT and IDCT modules with 24-bit approximate adders. Our simulation results showed that for $k \geq 16$, ApproxADD$v2$ provides acceptable image quality (both subjectively and objectively). Further, as compared to existing AFA-based approximate adders, ApproxADD$v2$ provides better IQA metrics. As a whole, ApproxADDs can be preferred over existing AFA-based approximate adders while designing circuits for current/future error-resilient applications.

REFERENCES

Richard P. Brent and H. T. Kung. 1982. A regular layout for parallel adders. *IEEE Trans. Comput.* C-31, 3 (1982), 260–264.

J. Han, C. Liu, and Fabrizio Lombardi. 2015. An analytical framework for evaluating the error characteristics of approximate adders. *IEEE Trans. Comput.* 5 (May 2015), 1268–1281.

CEVA. 2012. CEVA's DSP Cores. Retrieved from http://www.ceva-dsp.com/DSP-Cores.

V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. 2013. Analysis and characterization of inherent application resilience for approximate computing. In *Proceedings of the 50th ACM/EDAC/IEEE Design Automation Conference (DAC'13)*. 1–9.

S. Dutt, S. Nandi, and G. Trivedi. 2016a. A comparative survey of approximate adders. In *Proceedings of the 26th International Conference Radioelektronika (RADIOELEKTRONIKA'16)*. 61–65.

S. Dutt, H. Patel, S. Nandi, and G. Trivedi. 2016b. Exploring approximate computing for yield improvement via re-design of adders for error-resilient applications. In *Proceedings of the 29th International Conference on VLSI Design and 15th International Conference on Embedded Systems (VLSID'16)*. 134–139.

H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. 2011. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA'11)*. 365–376.

A. Gupta, D. E. Culler, and J. P. Singh. 1997. *Parallel Computer Architecture: A Hardware/Software Approach (1st ed.)*. Morgan Kaufmann.

V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy. 2013. Low-power digital signal processing using approximate adders. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Sys.* 32, 1 (Jan. 2013), 124–137.

Mark D. Hill and Christos Kozyrakis. 2012. Outbrief of DARPA/ISAT Workshop: Advancing Computer Systems without Technology Progress. Retrieved from http://www.sigarch.org/2012/07/28/outbrief-of-darpaisat-workshop-advancing-computer-systems-without-technology-progress/.

HSPICE. 2013. The Gold Standard for Accurate Circuit Simulation. Retrieved from http://www.synopsys.com/Tools/Verification/AMSVerification/CircuitSimulation/HSPICE/Pages/default.aspx.

Honglan Jiang, Jie Han, and Fabrizio Lombardi. 2015. A comparative review and evaluation of approximate adders. In *Proceedings of the ACM 25th Edition on Great Lakes Symposium on VLSI (GLSVLSI'15)*. 343–348.

A. Kabbani. 2008. Modeling and optimization of switching power dissipation in static CMOS circuits. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'08)*. 281–285.

A. B. Kahng and Seokhyeong Kang. 2012. Accuracy-configurable adder for approximate arithmetic designs. In *Proceedings of the 49th ACM/EDAC/IEEE Design Automation Conference (DAC'12)*. 820–825.

Y. Kim, Y. Zhang, and P. Li. 2013. An energy efficient approximate adder with carry skip for error resilient neuromorphic VLSI systems. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'13)*. 130–137.

Peter M. Kogge and Harold S. Stone. 1973. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Trans. Comput.* C-22, 8 (1973), 786–793.

M. Kudelka. 2012. Image quality assessment. In *Proceedings of the 21st Annual Conference of Doctoral Students (WDS'12)*. 94–99.

J. Liang, J. Han, and F. Lombardi. 2013. New metrics for the reliability of approximate and probabilistic adders. *IEEE Trans. Comput.* 62, 9 (Sep. 2013), 1760–1771.

Shih Lien Lu. 2004. Speeding up processing with approximation circuits. *IEEE Trans. Comput.* 37, 3 (Mar. 2004), 67–73.

H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas. 2010. Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications. *IEEE Trans. Circ. Syst. I* 57, 4 (Apr. 2010), 850–862.

MATLAB. 2015. Version 8.5 (R2015a). Retrieved from https://in.mathworks.com/products/matlab.html.

Mark Miller. 2005. Nanometer Yield Enhancement Begins in the Design Phase. Retrieved from http://electronicdesign.com/products/nanometer-yield-enhancement-begins-design-phase.

Sparsh Mittal. 2016. A survey of techniques for approximate computing. *Comput. Surv.* 48, 4 (Mar. 2016), 62:1–62:33.

Behrooz Parhami. 2010. *Computer Arithmetic: Algorithms and Hardware Designs (2nd ed.)*. Oxford University Press, UK.

Ramjee Prasad and Albena Mihovska. 2009. *New Horizons in Mobile and Wireless Communications: Radio Interfaces (1st ed.)*. Artech House.

PTM. 2007. Predictive Technology Model (PTM). Retrieved from http://ptm.asu.edu/modelcard/32nm_MGK.pm.

Jan Rabaey. 2009. *Low Power Design Essentials (1st ed.)*. Springer.

Doochul Shin and S. K. Gupta. 2008. A re-design technique for datapath modules in error tolerant applications. In *Proceedings of the 17th Asian Test Symposium (ATS'08)*. 431–437.

J. Sklansky. 1960. Conditional-sum addition logic. *IRE Trans. Electr. Comput.* EC-9, 2 (Jun. 1960), 226–231.

L. B. Soares, S. Bampi, and E. Costa. 2015. Approximate adder synthesis for area- and energy-efficient FIR filters in CMOS VLSI. In *Proceedings of the IEEE 13th International New Circuits and Systems Conference (NEWCAS'15)*. 1–4.

Ivan Sutherland, Bob Sproull, and David Harris. 1999. *Logical Effort: Designing Fast CMOS Circuits*. Morgan Kaufmann.

Tanner. 2015. Tanner S-Edit Schematic Capture. Retrieved from https://www.mentor.com/tannereda/s-edit.

USC-SIPI. 1977. The USC-SIPI Image Database. Retrieved from http://sipi.usc.edu/database/database.php.

A. K. Verma, P. Brisk, and P. Ienne. 2008. Variable latency speculative addition: A new paradigm for arithmetic circuit design. In *Proceedings of the Design, Automation & Test in Europe (DATE'08)*. 1250–1255.

Zhou Wang and Alan C. Bovik. 2006. *Modern Image Quality Assessment*. Morgan & Claypool.

Neil Weste and David Harris. 2010. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley.

X. Yang, Y. Xing, F. Qiao, Q. Wei, and H. Yang. 2016. Approximate adder with hybrid prediction and error compensation technique. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'16)*. 373–378.

Ning Zhu, Wang Ling Goh, and Kiat Seng Yeo. 2009. An enhanced low-power high-speed adder for error-tolerant application. In *Proceedings of the 12th International Symposium on Integrated Circuits (ISIC'09)*. 69–72.

Ning Zhu, Wang Ling Goh, Weija Zhang, Kiat Seng Yeo, and Zhi Hui Kong. 2010. Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing. *IEEE Trans. VLSI Syst.* 18, 8 (Aug. 2010), 1225–1229.

Received December 2016; revised May 2017; accepted July 2017