

# Systematic Synthesis of Approximate Adders and Multipliers with Accurate Error Calculations<sup>\*</sup>

Mohammadreza Esmali Nojehdeh<sup>a</sup>, Mustafa Altun<sup>a</sup>

<sup>a</sup>Department of Electronics and Communication Eng., Istanbul Technical University, Maslak, Istanbul, Turkey 34469

## ARTICLE INFO

### Keywords:

Approximate computing  
Logic synthesis  
Circuit design  
Ripple-carry adder  
Wallace-tree multiplier  
Neural network.

## ABSTRACT

In this study, we perform logic synthesis and area optimization of approximate ripple-carry adders and Wallace-tree multipliers with a given error constraint. We first implement approximate 1-bit adders having different error rates as building blocks of the proposed multi-bit adders and multipliers. In implementations, we exploit offsetting errors in carry and sum outputs of the adders. Also we take into account the probability of occurrence of input assignments. Using the implemented 1-bit adders, we systematically synthesize multi-bit adders and multipliers proceeding from the least to the most significant bits. We design the ripple-carry adders such that their successive 1-bit approximate adders cannot produce build-up errors. We design the Wallace-tree multipliers by considering the fact that their building blocks of 1-bit adders might have different probabilities of occurrence for different input assignments. As a result, the proposed adders and multipliers, implemented using the Cadence Genus tool with TSMC 0.18  $\mu\text{m}$  CMOS technology, offer in average a 25% smaller circuit area, and correspondingly power consumption, compared to the circuits proposed in the literature by satisfying the same error constraint. We also evaluate the adders and multipliers in image processing applications as well as within artificial neural networks.

## 1. Introduction

Approximate computing is used for area, power, and energy improvement, targeting applications not strictly requiring high accuracy including image processing and learning applications. Since arithmetic operations are the core of these applications, designing approximate adders and multipliers is crucial and extensively studied in the literature, especially in the last decade [1, 5, 6, 7, 8, 9, 13, 16, 18, 19, 28]. This study also focuses on the implementation of approximate arithmetic blocks in logic and circuit design levels. Considering their area and power efficiency among different adder and multiplier architectures, ripple-carry adders and Wallace-tree multipliers are selected for synthesis with an aim of optimizing the circuit area to satisfy a given error constraint.


Examining the related studies on approximate ripple-carry adders, we see a common tendency of assuming that the more erroneous outputs (Sum and Carry) we have, the less accurate our designs are [1, 6, 28]. This assumption neglects offsetting errors, errors in different outputs fully or partially cancelling each other. Motivated by this, we first design 1-bit full adders having offsetting errors in "Sum" and "Carry". We show that adding an error to an output might result in better accuracy and smaller area. Similarly in higher level, we design the ripple-carry adder such that two successive approximate 1-bit full adders cannot produce build-up errors or errors in different outputs cancelling each other. For example, if a 1-bit adder has an erroneous output of logic 1, expected as logic 0, we guarantee that the neighbour adder's output can be error free or it produces error with a logic 0 output, expected as logic 1.

In the literature, approximate ripple-carry adders are designed in two ways: 1) implementing 1-bit full adders in transistor level and then constructing a ripple-carry adder; 2) directly implementing a ripple-carry adder in logic level with synthesis tools. For the first way, approximate 1-bit adders are commonly derived from conventional mirror adders and XOR/XNOR based adders, by removing transistors and/or replacing some parts of the adders with smaller circuitries [1, 5, 6, 28]. In these studies, error dependencies in terms of offsetting and build-up errors are not considered. Also the implementation technique is not systematic, mostly based on the designer's experience and intuition. In the second way, the introduced tools are general purpose tools, not specifically for ripple-carry adders [3, 17, 23, 24, 25, 27, 30]. Also since finding near-optimal solutions needs much more time compared to the conventional non-approximate synthesis tools, these tools generally suffer from large runtimes. For instance, the method in [25] implements a 32-bit ripple-carry adder with 10% area saving, whereas truncation and the proposed methods yield 25% and 32% area savings, respectively for the same worst-case error value. If an average error value was used, then the results would be even worse with impractically high runtimes.

Motivated by the drawbacks of transistor and logic level approximation methods, we propose a systematic synthesis technique based on a new error calculation method. Our synthesis method is particularly for adders; it is fast, accurate, and scalable.

Along with ripple-carry adders, we implement Wallace-tree multipliers that consist of three stages: partial product generation, accumulation of partial product, and adding final results. In [15] approximate  $2 \times 2$  multiplier is introduced to generate partial products and exact adders for the accumulation tree. In [16], a new approximate adder is recommended for product accumulation where accurate adders

<sup>\*</sup>This work is supported by the TUBITAK-1001 project #117E078.

 nojehdeh@itu.edu.tr (M. Esmali Nojehdeh); altunmus@itu.edu.tr (M. Altun)

are exploited to recover error in final results. To speed up product accumulation and reduce the tree size, compressors are used in [16, 26]. Two approximate 4-2 compressor is proposed in [18] and relatively four different multipliers are introduced. Also by modifying this compressor with error recovery, a new multiplier is proposed in [7]. Additionally, in some studies truncation and rounding methods are exploited in the least significant columns of partial products [14, 22]. In [4], a bit-width aware multiplication algorithm is proposed with a carry-in prediction technique that improves the critical path of the multiplier. Also in [19], different approximate adders and multipliers library is introduced. All of these circuits are derived from conventional multipliers and adders based on multi-objective Cartesian genetic programming (CGP). Additionally, reconfigurable approximate arithmetic units are also proposed. Reconfiguration-oriented adder is proposed in [29]; bit wise multipliers are proposed in [10, 20] where detecting leading one block provides smaller cost in terms of area and power for targeted error values. Although reconfigurable circuits are adjustable for different error values, they need extra control blocks that increases their area and power consumption. Since we focus on achieving minimum area and power as opposed to reconfigurability, these studies are not considered in this work.

Different from the mentioned studies, we propose approximate full-adder and half-adder in both accumulation of partial product and in final result summation. Our design strategy is based on the occurrence probabilities of input assignments of 1-bit adders. We assign higher error rates for the input assignments having lower probabilities. As an example, consider a circuit having two inputs, and consider two different scenarios. For the first one, both inputs take the value of logic 1 and 0 with equal probabilities of 1/2. For the second scenario, both inputs take the value of logic 1 and 0 with probabilities of 1/4 and 3/4, respectively. We comment that for these two scenarios, area optimization techniques for a given error constraint should be different. While, an error corresponding to each input assignment equally contributes to the total error for the first scenario, this is different for the second one. Therefore, in our example we have more erroneous outputs corresponding to the input assignments with 1/4 probabilities than those for 3/4 probabilities. This leads us to synthesize full and half adders as the building blocks of the Wallace-tree multiplier based on the probability of inputs. To our knowledge our synthesis technique offers the smallest area in average compared to those in the literature, by satisfying the same error constraint. Also, our systematic synthesis technique is quite fast yet accurate in terms of error calculations.

The paper is organized as follows. In Section 2 and Section 3, we present our approximate synthesis methodologies for ripple-carry adders and Wallace-tree multipliers, respectively. We perform area optimization satisfying a given error objective. In Section 4, we present experimental results to compare the proposed adders and multipliers with their predecessors. We also evaluate them in image processing applications as well as in a learning application using a fully-

connected neural network. Finally in Section 5, we present conclusions and discuss future directions.

## 2. Ripple-carry adder design

Our synthesis techniques consists of two steps, given in the following two subsections. In the first step, we create a library of approximate 1-bit full adders with different error rates. In the second step, we systematically synthesize an  $n$ -bit ripple-carry adder from the least to the most significant bits by using the obtained library; the adder satisfies the given error constraint with minimum area.

### 2.1. 1-bit full adder design

A 1-bit full adder has three binary inputs: A, B and C (Carry or Cin), and two binary outputs Cout and Sum that comprises a two digit binary number. We represent the expected and the real decimal values of the output as  $\hat{y}$  and  $y$ , respectively. Note that they are in the range of decimal  $[0 - 3]$ . As an error metric, we use total absolute error distance (TAED):

$$\text{TAED} = \sum_{i=0}^7 |y_i - \hat{y}_i| \quad (1)$$

where  $i$  refers to the  $i$ th input assignment of the truth table. For example, if both Cout and Sum are grounded for all inputs assignments, that is indeed truncation offering zero cost for circuit area, then  $\text{TAED} = 12$ . However, we show that the zero cost can be achieved with  $\text{TAED} = 4$ . We also synthesize adders with  $\text{TAED} = 1$ ,  $\text{TAED} = 2$ , and  $\text{TAED} = 3$ . Our synthesis technique is based on exploiting offsetting errors in Sum and Cout. As a motivating example, consider three different approximate adders having an erroneous output for a certain input assignment. For the first adder there is  $0 \rightarrow 1$  error in Sum, so  $\text{TAED} = 1$ ; for the second one there is  $1 \rightarrow 0$  error in Cout, so  $\text{TAED} = 2$ ; and for the third one that is the proposed one, both of the errors occurs, so  $\text{TAED} = 1$  since simultaneous  $0 \rightarrow 1$  and  $1 \rightarrow 0$  errors occurring in Sum and Cout for the same input assignment results in a change of 1 in TAED. Among these three adders, the third one has much smaller area due to simultaneous minimization in Sum and Cout.

The main idea behind the proposed design approach is using offsetting errors  $0 \rightarrow 1$  and  $1 \rightarrow 0$  to the same input assignments. There are 6 input assignments in an exact adder (EXAD) to be used for our approximation technique (6 of 8 inputs has different Sum and Cout values). By considering these assignments and selecting the solutions offering minimum literal costs in sum-of-products (SOP) expressions, we present four different approximate adders APAD1, APAD2, APAD3, and APPAD4 respectively having TAED values of 1, 2, 3 and 4. Examples are given as truth tables in Table 1. As follows we elaborate on each adder type.

**Logic synthesis of APAD1, TAED = 1:** Offsetting errors in

Inputs A B Cin	Adder Type																		
	EXAD			APAD1				APAD2				APAD3				APAD4			
A B Cin	Cout	Sum	Decimal	Cout	Sum	Error	Decimal	Cout	Sum	Error	Decimal	Cout	Sum	Error	Decimal	Cout	Sum	Error	Decimal
0 0 0	0	0	0	0✓	0✓	0	0	0✓	0✓	0	0	0✓	0✓	0	0	0✓	0✓	0	0
0 0 <u>1</u>	0	1	1	0✓	1✓	0	1	0✓	1✓	0	1	0✓	1✓	0	1	<u>0</u> ✓	0X	<u>-1</u>	0
0 1 <u>0</u>	0	1	1	<u>1</u> X	0X	<u>+1</u>	2	0✓	1✓	0	1	0✓	1✓	0	1	0✓	1✓	0	1
0 1 <u>1</u>	1	0	2	1✓	0✓	0	2	<u>0</u> X	1X	<u>-1</u>	1	<u>0</u> X	1X	<u>-1</u>	1	<u>0</u> X	1X	<u>-1</u>	1
1 0 <u>0</u>	0	1	1	0✓	1✓	0	1	<u>1</u> X	0X	<u>+1</u>	2	<u>1</u> X	0X	<u>+1</u>	2	<u>1</u> X	0X	<u>+1</u>	2
1 0 1	1	0	2	1✓	0✓	0	2	1✓	0✓	0	2	1✓	0✓	0	2	1✓	0✓	0	2
1 1 <u>0</u>	1	0	2	1✓	0✓	0	2	1✓	0✓	0	2	1✓	1X	<u>+1</u>	3	<u>1</u> ✓	1X	<u>+1</u>	3
1 1 1	1	1	3	1✓	1✓	0	3	1✓	1✓	0	3	1✓	1✓	0	3	1✓	1✓	0	3

Table 1

Examples of Truth Tables of the Proposed Approximate Adders APAD's.

both Sum and Cout for one input assignment. There are  $\binom{6}{1}$  candidates for synthesis, and 2 of them have the minimum literal cost. Fig. 1 (a) shows one of them with  $Cout = B + ACin$ ;  $Sum = ABCin + AB\overline{Cin} + \overline{A}BCin$ .

**Logic synthesis of APAD2, TAED = 2:** Offsetting errors in both Sum and Cout for two input assignments. There are  $\binom{6}{2}$  candidates for synthesis, and 3 of them have the minimum literal cost. Fig. 1 (b) shows one of them with  $Cout = A$ ;  $Sum = \overline{A}B + ACin + BCin$ .

**Logic synthesis of APAD3, TAED = 3:** Since we reached a literal cost of 1 for Cout in APAD2, no further 0-1 transitions are preferred for Cout. However, for Sum we use one more error. As a result, offsetting errors in both Sum and Cout for two input assignments, and an error in Sum for one input assignment return TAED = 3. There are  $\binom{6}{2}\binom{4}{1}$  candidates for synthesis, and 9 of them have the minimum literal cost. Fig. 1 (c) shows one of them with  $Cout = A$ ;  $Sum = \overline{A}Cin + B$ .

**Logic synthesis of APAD4, TAED = 4:** Similar to APAD3 synthesis, we use offsetting errors in both Sum and Cout for two input assignment. Additionally, we inject errors in Sum for two input assignments, so TAED = 4. There are  $\binom{6}{2}\binom{4}{2}$  candidates for synthesis, and 9 of them have the minimum literal cost. Fig. 1 (d) shows one of them with  $Cout = A$ ;  $Sum = B$ .

## 2.2. n-bit ripple-carry adder design

We design the ripple-carry adder such that two successive approximate 1-bit full adders cannot produce build-up errors. For example, if a 1-bit adder has an erroneous output of logic 1, expected as logic 0, we guarantee that the neighbour adder's output can be error free or it produces an offsetting error with a logic 0 output, expected as logic 1. The following lemma gives necessary and sufficient conditions to eliminate build-up errors.

**Lemma 1.** Consider a ripple-carry adder having different 1-bit approximate adders. Build-up errors in successive adders of any type are eliminated if and only if all of the following requirements are met:

1. For all input assignments causing error,  $Cout = \overline{Cin}$ ;
2. For all input assignments causing a positive error that increases the expected output, all corresponding Cout's should be the same ; and
3. For all input assignments causing a negative error, decreasing the expected output, all corresponding Cout's should be the same.

*Proof.* The proof is by contradiction. For the first requirement, if a 1-bit adder has an assignment causing error with  $Cout=C$ , then successively using this adder would result in build-up errors. Again violating the second or the third requirement with even satisfying the first one causes a build-up error. Finally, satisfying these three requirements is sufficient to prevent build-up errors.  $\square$

To elucidate this lemma consider a 2-bit adder, where the least significant bit adder is APAD4 and the second adder can be any of proposed APADs. Let us investigate all possible inputs and their inexact results. Start by 001 as an input of APAD4, that results in 0s for both Cout and Sum. According to Table 1 this result is smaller than the exact result, so we name it as negative error. Since Cout of APAD4 is Cin of the second adder, we investigate inputs with Cin=0 for the second adder. According to Table 1 when Cin is 0, the generated results are exact results or positive errors. i.e if APAD4 generates negative error, it is impossible to have a positive error for the second adder. Next, consider the other input 011 causing negative error; same considerations are applicable for this case too. For positive error cases, inputs are 100 and 110; for both cases Cout value is 1. According to Table 1, if there is a positive error, Cin is always 0. Therefore two consecutive positive error is impossible too.

To satisfy Lemma 1, we shrink our 1-bit approximate adder library. Initially, it consists of 2 APAD1, 3 APAD2, 9 APAD3, and 9 APAD4, and it becomes 2 APAD1, 2 APAD2, 2 APAD3, and 2 APAD4, all satisfying Lemma 1 with  $Cin = 0$  for positive errors,  $Cin = 1$  for negative errors, and  $Cout = \overline{Cin}$ . Note that for each APAD type, we have two options with identical area and error performances. For simplicity we use the

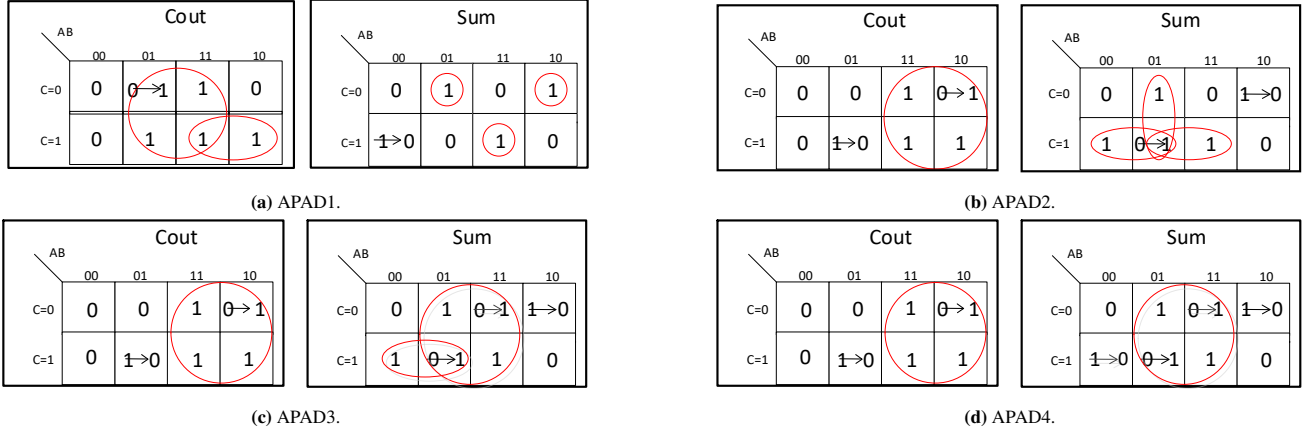


Figure 1: Karnaugh Maps of APADs.

adders given in Table 1, where all of proposed adders, satisfying Lemma 1 for all experiments.

In our synthesis technique, starting from the least to the most significant bit, we benefit the ordering of APAD4-APAD3-APAD2-APAD1-EXAD, where this array is justified with the following lemma.

**Lemma 2.** *Consider two successive 1-bit adders in a ripple-carry adder. To achieve minimum area with a given error constraint, the one closer to the least significant bit should have a larger or an equal TAED value compared to the one closer to the most significant bit.*

*Proof.* By contradiction, assume that the statement is wrong. By interchanging the two adders, we achieve a smaller error with the same area. However, the minimum area should have a negative correlation with the given error constraint. There is a contradiction.  $\square$

To understanding this lemma consider two scenarios for 2-bit adder. In the first scenario the least significant bit adder is APAD4 and the second bit adder is APAD1. In the second scenario the least significant bit adder is APAD1 and the second bit adder is APAD4. Both cases hold same area, but the first scenario results in smaller error.

In our synthesis technique, we use average absolute error distance (AAED) that is obtained with dividing TAED by the number of input assignments. For example AAED values of APAD4, APAD3, APAD2, and APAD1 are 4/8, 3/8, 2/8, and 1/8, respectively. We model AAED value and named it estimated average absolute error distance (EAAED). For an  $n$ -bit ripple-carry adder:

$$EAAED = \sum_{i=0}^{n-1} E_i 2^{i-1} \quad (2)$$

where  $E_i$  represents the error contribution of the  $i$ th 1-bit adder from the least significant bit.

$$E_i = \frac{2}{3} \sum_{a \in \{-1, 0, 1\}} \sum_{b \in \{-1, 1\}} P(i-1 : a, i : b) |0.5a + b| \quad (3)$$

where  $a$  and  $b$  represent the error values of the  $(i-1)$ th and  $i$ th 1-bit adders, respectively. Since, Equation 3 gives the error contribution of the  $i$ th adder,  $b = 0$  case, no error in the  $i$ th adder, is excluded. In the equation,  $P$  represents a probability that the  $(i-1)$ th and  $i$ th adders have errors of  $a$  and  $b$ , respectively. The constant factor  $2/3$  is the ratio of the error contribution of the  $i$ th adder ( $X$ ) to the total error contribution of the  $i$ th and the  $(i-1)$ th adders ( $0.5X + X$ ). In a similar fashion,  $|0.5a + b|$  represents the total error distance caused by the  $(i-1)$ th and the  $i$ th adders. In calculating  $P$ 's we use conditional probability such that  $P(i-1 : a, i : b) = P(i-1 : a)P(i : b | i-1 : a)$ . The following example elucidates our calculation steps of  $E_i$  given in Equation 3.

**Example 1.** Calculate  $E_i$  if the  $(i-1)$ th and the  $i$ th adders are both APAD4.

Table 2 gives the calculations by using the truth table of APAD4, previously given in Table 1. There are six cases in Table 2 corresponding to six rows in the table for different assignments of  $a$  and  $b$ . For the first and the sixth cases  $P$ 's are zero, since two successive positive error or negative error is impossible. For the second case,  $P(i-1 : -1) = 2/8$  since APAD4 has 2 input assignments causing  $-1$  error among 8 total assignments. Additionally,  $P(i : +1 | i-1 : -1) = 2/4$  since  $-1$  error causes Cout = 0 for the  $(i-1)$ th adder, so the  $i$ th adder's  $C$  is logic 0 and it has 2 input assignments causing  $+1$  error among 4 total assignments with  $C = 0$ . A similar justification can be done for the fifth case. For the third and the fourth cases,  $P(i-1 : 0) = 4/8$  and since  $a = 0$ ,  $P(i : b | i-1 : 0) = P(i : b) = 2/8$ .

Table 3 gives  $E_i$  values for all different combinations of APAD's as the  $(i-1)$ th and the  $i$ th adders with satisfying Lemma 2. Since  $C = 0$  for the ripple-carry first adder, we can obtain  $E_1$  as 2/4 for APAD4 and APAD3, and 1/4 for APAD2 and APAD1 by using the truth tables in Table 1.

**Example 2.** Calculate EAAED of an 8-bit ripple-carry adder having APAD4-APAD4-APAD4-APAD4-APAD2-APAD1-EXAD-



$a$	$b$	$P = P(i-1 : a) \times P(i : b   i-1 : a)$	$ 0.5a + b $	$\sum_{b \in \{-1,1\}}$
-1	-1	$2/8 \times 0$	$3/2$	0
-1	+1	$2/8 \times 2/4$	$1/2$	$1/16$
0	-1	$4/8 \times 2/8$	1	$2/16$
0	+1	$4/8 \times 2/8$	1	$2/16$
+1	-1	$2/8 \times 2/4$	$1/2$	$1/16$
+1	+1	$2/8 \times 0$	$3/2$	0
$\frac{2}{3} \sum_{a \in \{-1,0,1\}} \sum_{b \in \{-1,1\}} P(i-1 : a, i : b)  0.5a + b  = 4/16$				

**Table 2**Calculation of  $E_i$  for Example 1.

$i-1$ $i$	APAD1	APAD2	APAD3	APAD4
APAD1	2.66/32	✗	✗	✗
APAD2	2.33/32	4.66/32	✗	✗
APAD3	2.33/32	4.33/32	6.66/32	✗
APAD4	2/32	4/32	6/32	8/32

**Table 3**Values of  $E_i$ 's for Different APAD Combinations.

EXAD 1-bit adders ordered from the least to the most significant bit.

With  $E_1 = 2/4$ , and using Table 3:  $EAAED = \frac{2}{4}2^0 + \frac{8}{32}(2^1 + 2^2 + 2^3) + \frac{4}{32}2^4 + \frac{2.33}{32}2^5 = 8.33$ .

Constructed on Lemma 2 and the proposed error calculation method summarized in Table 3, our synthesis technique consists of the following 5 steps.

1. Start with an exact ripple-carry adder consisting of EXAD's.
2. From the least to the most significant bit, replace EXAD's with APAD4's until the calculated error value is larger than the given target error value.
3. Repeat the second step for APAD3, APAD2, and APAD1 instead of APAD4, respectively, replacing the unchanged EXAD's. Save the solution.
4. Replace all APAD3's, APAD2's, and APAD1's with EXAD's in Step 3; replace the last APAD4 (most significant one) respectively with APAD3, APAD2, and APAD1; apply the second step with APAD3, APAD2 and APAD1 instead of APAD4. Save solutions.
5. Using area costs of APAD4, APAD3, APAD2, APAD1, and EXAD, select the best solution with minimum area cost.

Note that due to the essence of the proposed method, without using any error detection block that causes area overhead, build-up errors are fully eliminated. To elucidate our synthesis technique, we present an example.

**Example 3.** With a given target  $AAED = 3.9$ , synthesize an approximate 8-bit ripple-carry adder. Suppose that the 1-bit adders are implemented with a generic library consisting of NAND2 gates (4 transistors) and inverters (2 transistors); APAD4, APAD3, APAD2, APAD1, and EXAD has transistor costs of 0, 12, 20, 32, and 44, respectively.

Desired Error	Proposed Method			Exhaustive Search		
	Ripple-carry	Est. AAED	Time (s)	Ripple-carry	Exact AAED	Time (s)
1.5	E E E E E 2 4 4	1.5	.02	E E E E E 2 4 4	1.5	$>10^4$
2.9	E E E E 2 2 4 4	2.665	.022	E E E E 2 2 4 4	2.875	$>10^4$
4.5	E E E 1 2 4 4 4	4.165	.016	E E E 1 2 4 4 4	4.46	$>10^4$
7.5	E E E 3 4 4 4 4	7	.02	E E E 3 4 4 4 4	7.125	$>10^4$
18	E 1 2 4 4 4 4 4	16.66	.017	E 1 2 4 4 4 4 4	17.59	$>10^4$
48	2 4 4 4 4 4 4 4	48	.016	2 4 4 4 4 4 4 4	48	$>10^4$
64	4 4 4 4 4 4 4 4	64	.023	4 4 4 4 4 4 4 4	64	$>10^4$

**Table 4**

Synthesis of 8-bit Adders with the Proposed Synthesis Technique and Exhaustive Search.

Steps are shown in Fig. 2. In Step 4 check-marks are for satisfying given error and Lemma 2.

Examining our technique, we see that the second and the third steps constitute the core of it. The fourth step is a backtracking step to find other candidates for minimum area. In the fifth step, we have three solutions and the one with the smallest area cost wins. From our experiments, we see that the first solution is generally the best one. However, different area costs of 1-bit adders would change this. For example, suppose that area costs are 70, 45, 20, 10 and 0 for EXAD, APAD1, APAD2, APAD3, and APAD4, respectively. Synthesis results with these costs are given in Table 4 by comparing our synthesis technique with the exhaustive search technique for different AAED's.

The exhaustive search is done by checking all approximate and exact adder combinations satisfying Lemma 2 as well as by obtaining exact AAED values with testing all possible input combinations. The results prove the accuracy of our synthesis technique. Additionally, estimated AAED values are almost the same as the exact ones. For some cases, there are deviations, but it is expected since our calculation technique applies conditional probability to adder pairs by only considering the target adder and the previous adder. A fully exact calculation should consider all of the previous adders. However, this type of calculation would result in impractical run times as shown in Table 4.

Note that different technologies might result in different area sizes for APADs. However our synthesis algorithm is unrelated to the technology and always find near optimal area solutions.

### 3. Wallace-tree multiplier design

We implement Wallace-tree multipliers in a usual fashion by using 1-bit full adders, 1-bit half adders, and AND gates in three stages. This is shown in Fig. 3 for 4-bit inputs. In the first stage, the multiplier inputs  $a_0, a_1, a_2, a_3$  and  $b_0, b_1, b_2, b_3$  are first ANDed. Then the outputs of AND gates are fed to 1-bit adders. Considering that the multiplier

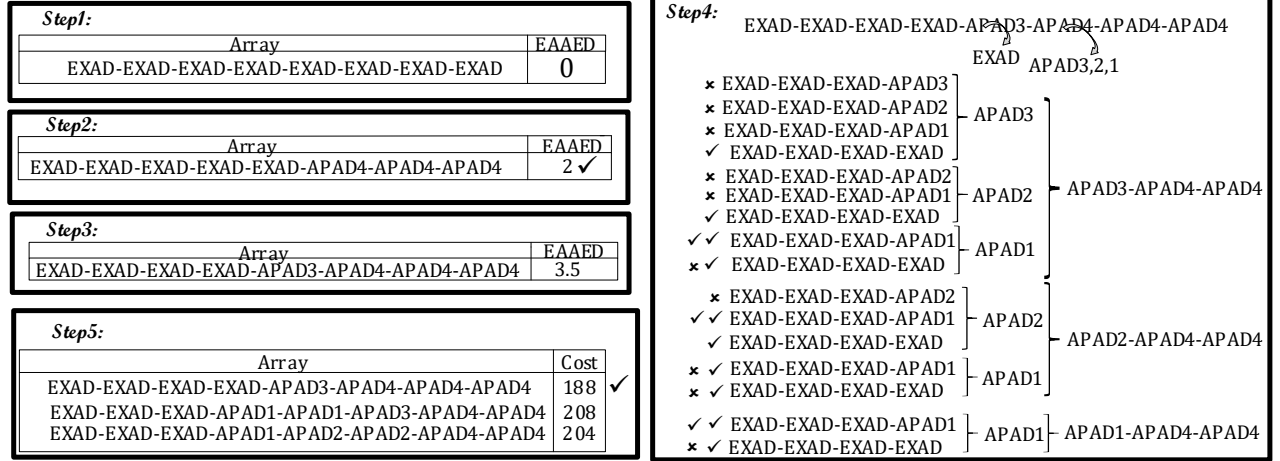


Figure 2: Demonstration of Steps for Example 3.

inputs take the values of logic 1 and 0 with equal probabilities of 1/2, the adder inputs take the values of logic 1 and 0 with probabilities of 1/4 and 3/4, respectively. Motivated by this fact, we prefer erroneous outputs corresponding to the input assignments with 1/4 probabilities. In other words, we synthesize full and half adders based on the occurrence probabilities of inputs. Therefore, we do not use the APAD's, previously used for the synthesis of ripple-carry adders in Section 2.

In designing 1-bit adders, another motivation is to achieve Cout = 0 that either converts the proceeding full adder to an half adder or rules out the proceeding half adder, without loss of accuracy. That is why we do not need approximate full adders in the second and the third stages; only approximate half adders are used in these stages. Similar to the ripple-carry adder synthesis in the previous section, we have two steps. In the first step, we create a library of approximate 1-bit full and half adders. In the second step, we systematically synthesize an n-bit Wallace-tree multiplier from the least to the most significant bits by using the library; the multiplier satisfies the given error constraint with minimum area. The following two subsections explain these two steps.

### 3.1. Design of 1-bit approximate full adder (APFA) and half adder (APHA)

**Logic synthesis of APFA, TAED = 1.375:** Table 5 shows the truth table of APFA. We use offsetting errors in both Sum and Cout for three input assignments having probabilities of 9/64, and to make Cout = 0, we make an error at Cout for the last input assignment with a probability of 1/64. As a result, Cout = 0 and Sum = A + B + Cin. The adder's TAED value is given by:

$$TAED = \sum_{i=0}^7 8 |y_i - \hat{y}_i| P_i \quad (4)$$

where  $P_i$  represents an occurrence probability of the  $i$ th input assignment. Using Table 5, we can find that TAED =  $8(3/64 + 3/64 + 3/64 + 2/64) = 1.375$ . Due to varied probabilities for input assignments, we achieve an adder that is much more efficient than the APAD's.

**Logic synthesis of APHA, TAED = 0.25, 1.34, 0.58:** Table 6 shows the truth table of APHA. We use offsetting errors in both Sum and Cout for the last input assignment with a probability of 1/16. As a result, Cout = 0 and Sum = A + B; total literal cost is 2. The adder's TAED value is given by:

$$TAED = \sum_{i=1}^4 4 |y_i - \hat{y}_i| P_i \quad (5)$$

where  $P_i$  represents an occurrence probability of the  $i$ th input assignment. In contrary to APFA's that are only used in the first stage, APHA's are used in all stages. Therefore we have different  $P_i$  values for three different cases as shown in Table 6. Case1 corresponds to the first stage. Case2 occurs when the inputs A and B of APHA are connected to Sum outputs of APFA. Case3 occurs when the input A of APHA is connected to the output of an AND gate, and the input B of APHA is connected to a Sum output of APFA. As a result, Case1, Case2, and Case3 have TAED values of 0.25, 1.34, and 0.58, respectively.

### 3.2. n-bit Wallace-tree multiplier design

The multiplier synthesis technique is more straightforward than ripple-carry adder, also unlike to adders, error calculation method is effectively accurate for proposed approximate multipliers. Restricted exploiting of APFA and APHA results only negative errors, thus summation of errors enable us to obtain exact error value. Additionally using APFA or APFA decreases the number of inputs of the pro-

Inputs			Adder Type				Error	
			EXAD		APFA		Error	Probability
A	B	Cin	Sum	Cout	Sum	Cout		
0	0	0	0	0	0✓	0✓	0	$3/4 \times 3/4 \times 3/4 = 27/64$
0	0	1	1	0	1✓	0✓	0	$3/4 \times 3/4 \times 1/4 = 9/64$
0	1	0	1	0	1✓	0✓	0	$3/4 \times 1/4 \times 3/4 = 9/64$
0	1	1	0	1	1✗	0✗	-1	$3/4 \times 1/4 \times 1/4 = 3/64$
1	0	0	1	0	1✓	0✓	0	$1/4 \times 3/4 \times 3/4 = 9/64$
1	0	1	0	1	1✗	0✗	-1	$1/4 \times 3/4 \times 1/4 = 3/64$
1	1	0	0	1	1✗	0✗	-1	$1/4 \times 1/4 \times 3/4 = 3/64$
1	1	1	1	1	1✓	0✗	-2	$1/4 \times 1/4 \times 1/4 = 1/64$

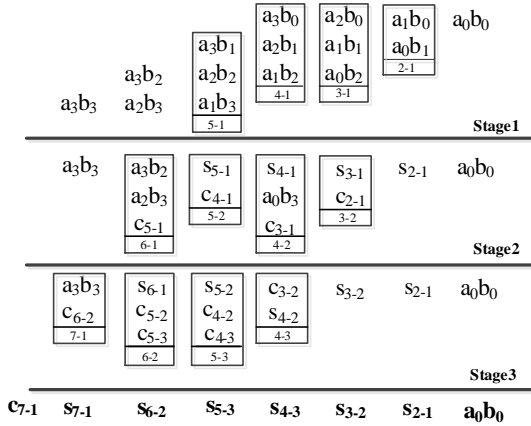
**Table 5**

Truth Table of the Proposed Approximate Full Adder APFA.

Inputs		Adder Type				Error			
		EXAD Sum Cout		APHA Sum Cout		Error	Probability		
							Case1 (Stage1)	Case2 (Stage2-3)	Case3 (Stage2-3)
0	0	0	0	0✓	0✓	0	$3/4 \times 3/4 = 9/16$	$27/64 \times 27/64 = 729/4096$	$3/4 \times 27/64 = 81/256$
0	1	1	0	1✓	0✓	0	$3/4 \times 1/4 = 3/16$	$27/64 \times 37/64 = 999/4096$	$3/4 \times 37/64 = 111/256$
1	0	1	0	1✓	0✓	0	$1/4 \times 3/4 = 3/16$	$27/64 \times 37/64 = 999/4096$	$1/4 \times 27/64 = 27/256$
1	1	0	1	1✗	0✗	-1	$1/4 \times 1/4 = 1/16$	$37/64 \times 37/64 = 1369/4096$	$1/4 \times 37/64 = 37/256$

**Table 6**

Truth Table of the Proposed Approximate Half Adder APHA.


**Figure 3:** 4×4 bit Exact Wallace-tree Multiplier.

ceeding adder by one, so only two successive approximate adders can be employed.

Similar for the ripple-carry adders, we again use AAED (average absolute error distance). This value is obtained with dividing TAED by the number of input assignments. For example AAED of APFA is  $1.375/8$ . Regarding to input probability and negative error inherent of APFA and APHA, n-bit multiplier AAED value is formulated as below.

$$AAED = \sum_i \sum_j AAED_{i-j} 2^{i-1} \quad (6)$$

where  $AAED_{i-j}$  represents the error contribution of the adder in the  $i$ th column and the  $j$ th row of the Wallace-tree structure. Check that the multiplier in Fig. 3 has 7 columns and 3 rows.

Our synthesis technique consists of the following 5 steps:

1. Start with an exact Wallace-tree multiplier.
2. Replace an exact adder having the smallest column and row numbers (column numbers are more significant) with an approximate adder. Calculate AAED.
3. Update the multiplier structure without loss of accuracy by converting full adders to half adders and/or ruling out half adders.
4. Repeat the second and the third steps until the calculated error value AAED is larger than the given target error value and store result.
5. Obtain the area cost of the multiplier by using the area costs AND gates, exact adders, and approximate adders.

To elucidate our synthesis technique, we present an example.

**Example 4.** With a given target  $AAED = 1$ , synthesize an approximate 4-bit×4-bit Wallace-tree multiplier. Suppose that the circuits are implemented with a generic library consisting of NAND2 gates (4 transistors) and inverters (2 transistors); AND2 gate, APHA, APFA, exact half adder, and exact full adder has transistor costs of 6, 8, 16, 14, and 44, respectively.

In the first step, we have an exact multiplier having 6 full adders, 6 half adders, and 16 AND2 gates as shown in Fig.

3. In the second step, we start with the half adder in the place 2-1, to be replaced by APHA ( $AAED_{2-1} = 0.25/4$ ). In the third step, we first rule out the half adder in 3-2 since  $c_{2-1} = 0$ , the half adder in 3-2 becomes  $s_{3-1}$  that also makes  $c_{3-2} = 0$ . Similarly, the half adder in 4-3 is ruled out. In the fourth step since AAED is smaller than the target error rate, we repeat the second and third steps.

We replace the full adder in 3-1 with APFA ( $AAED_{3-1} = 1.375/8$ ) that converts the full adder in 4-2 to an half adder. The total error is given by  $AAED = (0.25/4)2^1 + (1.375/8)2^2 = 0.8125$ . Since the next approximation in 4-1 would make AAED exceed the target error of 1, we stop here.

In the last step, since the final multiplier structure has 4 exact full adders, 3 exact half adders, 1 APFA, 1 APHA, and 16 AND2 gates, we achieve the total area cost of 338 (24% area saving).

## 4. Experimental results

All of the circuits are implemented in the same environment using the Cadence Genus tool with TSMC 0.18μm CMOS technology.

### 4.1. Area, power, delay, and energy versus average error

Table 7 evaluates the proposed 1-bit adders. While XOR/XNOR based adders from [1] and mirror adder based adders from [6] are synthesized in transistor level, the rest of the adders including the proposed ones, are obtained with logic synthesis algorithms using standard gate libraries. For comparison, among many different 1-bit adders in the literature, we consider the best and working ones. For example, since that INXA1 adder in [1] does not work properly, having stability problems, we do not consider it. Among the adders in [6], only AMA3 is selected for comparison since it performs much better compared to the other AMAs. Also AMA5 in this work is the same as the proposed APAD4, however design methodology of AMAs and our method is completely different. Note that APAD4 and AMA5 have zero area and power.

In Table 7 all of the proposed APADs are derived from an exact adder with aforementioned synthesis method. Since mirror based exact adders and XOR/XNOR based exact adders are built in transistor level with low power considerations, inevitably their area and power is smaller than those of the standard logic-level exact adder. According to this table for same TAED values, the proposed APADs achieve better performance for most of the cases.

Table 8 evaluates the proposed 8-bit ripple-carry adders compared to other methods. Area, power, delay and power-delay product ( $PDP$ ) results for given AAED values are given.

AAED values calculated by simulation for all input combinations ( $256 \times 256$ ), and the case with maximum area save for given AAED constraint preferred. For comparison, AMA3 and INXA3 selected from Table 7 base on their better performances. Also the Evoapprox adders in [19] are chosen among the logic-synthesis approximation methods, since the

library generated in this study covers all competitive adders. Among different adders introduced in [19], we opted for adders with maximum area saving, for given AAED value. By investigating the results in Table 8, we see that the proposed and Evoapprox adders come forward, however the introduced adders in [19] are limited to 8-bit due to long run-time problem. The proposed adders generally are the best in area; for other performances the adders are comparable. These results imply inefficiency of transistor-level synthesis methods.

A similar analysis is made for the proposed multipliers in comparison with the other Wallace-tree multipliers in Table 9. On account of the fact that the multipliers in [7, 18] are compressor based, their exact area values are smaller than adder based multipliers; exact 8-bit multiplier area for compressor based in [7, 18] and adder based in [19] are  $7348\mu m^2$  and  $8097\mu m^2$ , respectively. Although area values of compressor based exact multipliers are generally smaller, for small errors values it is not applicable to achieve approximate versions. According to Table 9, the proposed multipliers almost always hold the smallest area and delay among investigated studies.

### 4.2. Image processing: peak signal to noise ratio (PSNR) versus area saving

We use mean filter and bit-wise multiplication to evaluate the adders and the multipliers, respectively. With the same area savings, we compare the obtained images' PSNR values.

For the mean filter application, we first inject a Gaussian noise having a zero mean and a variance of 0.008. Then we apply mean filter using the 8-bit adders having 75% area savings. For some of the adders, 75% is not achievable, so we consider their maximum achievable values. The results are shown in Fig. 4. The PSNR value for the proposed adder is 37.6dB that is the highest one.

For the bit-wise multiplication, we use two images to obtain the original image. For this purpose, we use the approximate 8-bit×8-bit multipliers having 40% area savings. Again, for some of the multipliers, 40% is not achievable, so we consider their maximum achievable values. The results are shown in Fig. 5. The PSNR value for the proposed multiplier is 37.1dB that is the highest one.

### 4.3. Neural network: misclassification rate versus area saving

We realize a handwriting recognition neural network. A trained database PENDIGIT is used [2]. The network has 16-100-10 architecture having three layers. Due to neural network essence  $16 \times 100$  multipliers and  $17 \times 100$  adders in hidden layer, also  $100 \times 10$  multipliers and  $101 \times 10$  adders in output layer, are used to obtain the desired network. It must be considered that training is done with unnormalized inputs.

We only use 12-bit adders and 8-bit×8-bit multipliers having a certain area saving, and then check misclassification rates. Again for comparison, we select the best adders



Adder Type	Results					
	Area $\mu m^2$	Delay $\rho s$	Average Power $\mu w$	Average Energy $\rho w s$	Worst case Power $\mu w$	TAED
Exact Adder	148	1080	348	14.98	2932	0
APAD1	148	955	582	14.24	1470	1
APAD2	74	220	208	2.63	1060	2
APAD3	55	183	226	2.41	930	3
AXExact[1]	53	9600	154	602	1280	0
INXA3[1]	44	9500	133	358	649	2
INXA2[1]	48	2800	353	456	772	2
AMAccurate[6]	169	718	231	14.27	1680	0
AMA3[6]	58	1200	290	13	616	3
Logic1[12, 23]	235	1050	417	18.68	2430	1
Logic2[12, 23]	84	957	291	14.1	1260	2
Logic4[12, 23]	105	970	229	13.3	1800	4
Carving[21]	105	1029	233	13.06	1800	4
BLASYS [11]	64	947	717	10.98	1086	4

**Table 7**

1-Bit Adder Results.

Adder Type	AAED																			
	0.75				2.2				3.3				6.6				11			
	Area $\mu m^2$	Power $\mu w$	Delay $\eta s$	PDP $aJ$	Area $\mu m^2$	Power $\mu w$	Delay $\eta s$	PDP $aJ$	Area $\mu m^2$	Power $\mu w$	Delay $\eta s$	PDP $aJ$	Area $\mu m^2$	Power $\mu w$	Delay $\eta s$	PDP $aJ$	Area $\mu m^2$	Power $\mu w$	Delay $\eta s$	PDP $aJ$
AMA3[6]	925	622	2167	13.4	746	500	2263	11.3	746	500	2263	11.3	652	412	2368	9.7	558	303	2472	7.4
INXA3[1]	869	594	2539	15	784	523	2853	14.9	784	523	2853	14.9	699	448	3166	14.2	615	366	3480	12.7
Evoapprox[19]	850	561	1657	9.3	630	363	1694	6.1	608	357	1148	4	458	194	1736	3.4	392	161	1566	2.5
Truncation	982	637	2459	15.6	850	560	2204	12.3	718	453	1949	8.8	718	453	1949	8.8	586	349	1694	5.9
Proposed	834	542	1912	10.3	649	400	1657	6.6	571	341	1403	4.8	439	231	1148	2.6	370	187	1275	2.4

**Table 8**

8-Bit Adder Results.

and multipliers by considering Table 8 and Table 9. Table 10 shows the results. The proposed circuits certainly overwhelm the compared ones, especially for large area savings.

## 5. Conclusion

In this study, we perform area optimization techniques for approximate ripple-carry adders and Wallace-tree multipliers to satisfy a given error constraint. Our techniques are accurate and fast, in courtesy of the proposed error calculation techniques that consider error dependencies of building blocks of adders and multipliers as well as occurrence prob-

abilities of input assignments. As a future work, we intend to extend our synthesis methodology to be directly applicable for image processing and learning applications with satisfying a given PSNR and misclassification rate.

## References

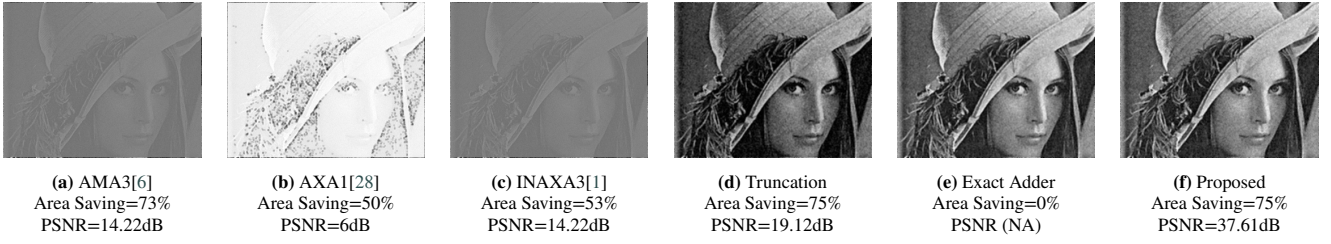
- [1] Almurib, H.A.F., Kumar, T.N., Lombardi, F., 2016. Inexact designs for approximate low power addition by cell replacement, in: 2016 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 660–665.
- [2] Alpaydin, E., Alimoglu, F., 1998. Pen-based recognition of handwritten

Multiplier Type	AAED																			
	1				5				10				20				50			
	Area $\mu m^2$	Power $\mu w$	Delay $\eta s$	PDP $aJ$	Area $\mu m^2$	Power $\mu w$	Delay $\eta s$	PDP $aJ$	Area $\mu m^2$	Power $\mu w$	Delay $\eta s$	PDP $aJ$	Area $\mu m^2$	Power $\mu w$	Delay $\eta s$	PDP $aJ$	Area $\mu m^2$	Power $\mu w$	Delay $\eta s$	PDP $aJ$
Momeni-1 [18]	7348	4797	6765	324	7225	4871	7139	347	7148	4668	7148	334	7065	4504	7374	332	6849	4232	7052	298
Momeni-2 [18]	7348	4797	6765	324	7244	4860	7090	344	6884	4460	7051	314	6789	4431	7120	315	6313	3770	7099	268
Minho [7]	7348	4797	6765	324	7147	4976	6890	343	7147	4976	6890	343	7094	4887	7150	349	6924	4680	6980	327
Evoapprox [19]	7922	5422	9162	496	7787	5522	9422	520	7297	5153	8899	458	6498	4342	10534	457	5701	4040	8150	329
Truncation	8006	5329	6469	346	7649	5363	5529	297	7338	5029	6760	340	7022	4398	6417	282	6222	4301	5370	230
Proposed	7721	5374	5628	302	7150	4992	5729	286	6959	4858	5573	270	6485	4410	5726	252	5761	3960	5450	216

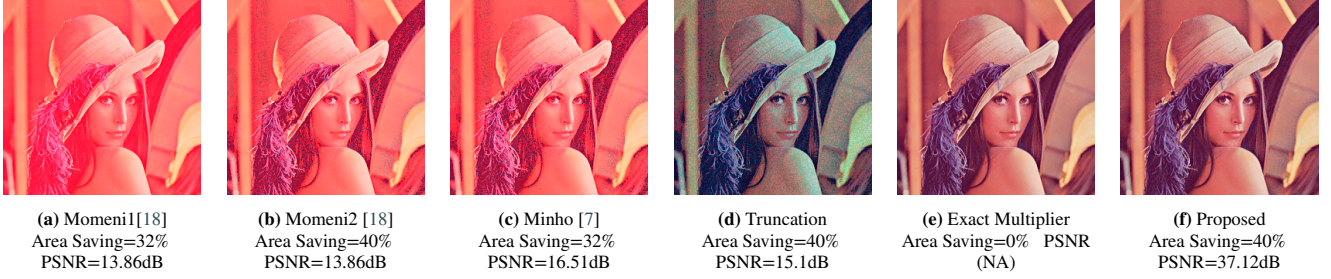
**Table 9**

8-Bit×8-Bit Multiplier Results.

## Approximate Adders & Multipliers



**Figure 4:** Mean Filter Results with Approximate 8-Bit Adders.



**Figure 5:** Results for Blending Two Images by Approximate 8-Bitx8-Bit Multipliers.

Multiplier Type	Adder Type	Area Saving				
		5%	13%	25%	34%	41%
Truncation	Truncation	3.03	3.0303	4.54	18.45	58.119
Evoapprox[19]	AMA3[6]	3.0017	3.259	3.259	3.5163	13.52
Proposed	Proposed	2.9445	3.0017	3.0303	3.6	3.6021

**Table 10**

Neural Network Misclassification Rates.

- ten digits data set. University of California, Irvine, Machine Learning Repository. Irvine: University of California .
- Bernasconi, A., Ciriani, V., 2014. 2-spp approximate synthesis for error tolerant applications, in: 2014 17th Euromicro Conference on Digital System Design, pp. 411–418. doi:10.1109/DSD.2014.21.
  - Bhardwaj, K., Mane, P.S., Henkel, J., 2014. Power- and area-efficient approximate wallace tree multiplier for error-resilient systems, in: Fifteenth International Symposium on Quality Electronic Design, pp. 263–269. doi:10.1109/ISQED.2014.6783335.
  - Gupta, V., Mohapatra, D., Park, S.P., Raghunathan, A., Roy, K., 2011. Impact: Imprecise adders for low-power approximate computing, in: IEEE/ACM International Symposium on Low Power Electronics and Design, pp. 409–414. doi:10.1109/ISLPED.2011.5993675.
  - Gupta, V., Mohapatra, D., Raghunathan, A., Roy, K., 2013. Low-power digital signal processing using approximate adders. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 32, 124–137. doi:10.1109/TCAD.2012.2217962.
  - Ha, M., Lee, S., 2017. Multipliers with approximate 4-2 compressors and error recovery modules, pp. 1–1. doi:10.1109/LES.2017.2746084.
  - Han, J., Orshansky, M., 2013. Approximate computing: An emerging paradigm for energy-efficient design, in: 2013 18th IEEE European Test Symposium (ETS), pp. 1–6. doi:10.1109/ETS.2013.6569370.
  - Hanif, M.A., Hafiz, R., Hasan, O., Shafique, M., 2017. Quad: Design and analysis of quality-area optimal low-latency approximate adders, in: Proceedings of the 54th Annual Design Automation Conference 2017, ACM, New York, NY, USA. pp. 42:1–42:6. URL: <http://doi.acm.org/10.1145/3061639.3062306>, doi:10.1145/3061639.3062306.
  - Hashemi, S., Bahar, R.I., Reda, S., 2015. Drum: A dynamic range unbiased multiplier for approximate applications, in: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, IEEE Press, Piscataway, NJ, USA. pp. 418–425. URL: <http://dl.acm.org/citation.cfm?id=2840819.2840878>.

- Hashemi, S., Tann, H., Reda, S., 2018. Blasys: Approximate logic synthesis using boolean matrix factorization, in: Proceedings of the 55th Annual Design Automation Conference, ACM, New York, NY, USA. pp. 55:1–55:6. URL: <http://doi.acm.org/10.1145/3195970.3196001>, doi:10.1145/3195970.3196001.
- Ichihara, H., Inaoka, T., Iwagaki, T., Inoue, T., 2015. Logic simplification by minterm complement for error tolerant application, in: 2015 33rd IEEE International Conference on Computer Design (ICCD), pp. 94–100. doi:10.1109/ICCD.2015.7357089.
- Jiang, H., Han, J., Qiao, F., Lombardi, F., 2016. Approximate radix-8 booth multipliers for low-power and high-performance operation. IEEE Transactions on Computers 65, 2638–2644. doi:10.1109/TC.2015.2493547.
- King, E.J., Swartzlander, E.E., 1997. Data-dependent truncation scheme for parallel multipliers, in: Conference Record of the Thirty-First Asilomar Conference on Signals, Systems and Computers (Cat. No.97CB36136), pp. 1178–1182 vol.2. doi:10.1109/ACSSC.1997.679090.
- Kulkarni, P., Gupta, P., Ercegovic, M., 2011. Trading accuracy for power with an underdesigned multiplier architecture, in: 2011 24th International Conference on VLSI Design, pp. 346–351. doi:10.1109/VLSID.2011.51.
- Liu, C., Han, J., Lombardi, F., 2014. A low-power, high-performance approximate multiplier with configurable partial error recovery, in: Proceedings of the Conference on Design, Automation & Test in Europe, European Design and Automation Association, 3001 Leuven, Belgium, Belgium. pp. 95:1–95:4. URL: <http://dl.acm.org/citation.cfm?id=2616606.2616722>.
- Miao, J., Gerstlauer, A., Orshansky, M., 2013. Approximate logic synthesis under general error magnitude and frequency constraints, in: 2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 779–786. doi:10.1109/ICCAD.2013.6691202.
- Momeni, A., Han, J., Montuschi, P., Lombardi, F., 2015. Design and analysis of approximate compressors for multiplication. IEEE Transactions on Computers 64, 984–994. doi:10.1109/TC.2014.2308214.
- Mrazek, V., Hrbacek, R., Vasicek, Z., Sekanina, L., 2017. Evoapproxsb: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods, in: Design, Automation Test in Europe Conference Exhibition (DATE), 2017, pp. 258–261. doi:10.23919/DATE.2017.7926993.
- Saadat, H., Bokhari, H., Parameswaran, S., 2018. Minimally biased

- multipliers for approximate integer and floating-point multiplication. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 2623–2635. doi:10.1109/TCAD.2018.2857262.
- [21] Scarabottolo, I., Ansaloni, G., Pozzi, L., 2018. Circuit carving: A methodology for the design of approximate hardware, in: 2018 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 545–550. doi:10.23919/DATE.2018.8342067.
  - [22] Schulte, M.J., Swartzlander, E.E., 1993. Truncated multiplication with correction constant [for dsp], in: Proceedings of IEEE Workshop on VLSI Signal Processing, pp. 388–396. doi:10.1109/VLSISP.1993.404467.
  - [23] Shin, D., Gupta, S.K., 2010. Approximate logic synthesis for error tolerant applications, in: Proceedings of the Conference on Design, Automation and Test in Europe, European Design and Automation Association, 3001 Leuven, Belgium, Belgium. pp. 957–960.
  - [24] Venkataramani, S., Roy, K., Raghunathan, A., 2013. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits, in: Proceedings of the Conference on Design, Automation and Test in Europe, EDA Consortium, San Jose, CA, USA. pp. 1367–1372. URL: <http://dl.acm.org/citation.cfm?id=2485288.2485615>.
  - [25] Venkataramani, S., Sabne, A., Kozhikkottu, V., Roy, K., Raghunathan, A., 2012. Salsa: Systematic logic synthesis of approximate circuits, in: Proceedings of the 49th Annual Design Automation Conference, ACM, New York, NY, USA. pp. 796–801. doi:10.1145/2228360.2228504.
  - [26] Wang, Z., Jullien, G.A., Miller, W.C., 1995. A new design technique for column compression multipliers, pp. 962–970. doi:10.1109/12.403712.
  - [27] Wu, Y., Qian, W., 2016. An efficient method for multi-level approximate logic synthesis under error rate constraint, in: Proceedings of the 53rd Annual Design Automation Conference, ACM, New York, NY, USA. pp. 128:1–128:6. URL: <http://doi.acm.org/10.1145/2897937.2897982>, doi:10.1145/2897937.2897982.
  - [28] Yang, Z., Jain, A., Liang, J., Han, J., Lombardi, F., 2013. Approximate xor/xnor-based adders for inexact computing, in: 2013 13th IEEE International Conference on Nanotechnology (IEEE-NANO 2013), pp. 690–693. doi:10.1109/NANO.2013.6720793.
  - [29] Ye, R., Wang, T., Yuan, F., Kumar, R., Xu, Q., 2013. On reconfiguration-oriented approximate adder design and its application, in: 2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 48–54. doi:10.1109/ICCAD.2013.6691096.
  - [30] Zou, C., Qian, W., Han, J., 2015. Dpals: A dynamic programming-based algorithm for two-level approximate logic synthesis, in: 2015 IEEE 11th International Conference on ASIC (ASICON), pp. 1–4. doi:10.1109/ASICON.2015.7516961.