A Case Study of Analyzing Imprecise Adders using BDDs

Cunxi Yu, Maciej Ciesielski
Department of Electrical Computer Engineering
University of Massachusetts, Amherst, MA, USA
Email: ycunxi@umass.edu, ciesiel@ecs.umass.edu

***Abstract*—Imprecise adders are implemented to improve the performance and power consumption of arithmetic circuits with forgivable inaccurate result. These types of designs are extensively used in digital computer systems for approximate computing. One of the challenges in designing imprecise adders is evaluation of output quality. Currently, the most popular technique to evaluate the approximate designs are random simulation and error estimation. These techniques cannot provide exact error analysis. In this paper, we present a formal approach of evaluating the imprecise adders based on BDDs. The proposed framework can measure *Exact error rate* (EER) of the designs. Additionally, we present a method for test generation (TG) of approximate adders, which can be used for implementing correlation logic. The proposed technique has been demonstrated using a number of imprecise adders (VLSA and ACA) with large number of errors (**$over1\ billion$**).**

I. Introduction

One of the greatest challenges of the modern VLSI designs is to reduce the high power consumption. Although circuit-level techniques, such as *dynamic voltage and frequency scaling* (DVFS), as well as *near-threshold* technique have proved effective in power reduction, they are still fundamentally limited by the critical path of the arithmetic circuits. To overcome this limitation, several approximate designs have been proposed that exploit a tradeoff between computation accuracy versus performance and power [1][2][3][4]. The approximate designs have proved to significantly improve energy efficiency for noise-tolerant applications. Especially, for applications related to artificial neural networks, approximate arithmetic designs generate sufficient accurate results compared to the traditional designs, which produce absolute accurate results [5][3][6].

One of the challenges for CAD tools in the approximate circuits design flow is evaluation of such designs. The key of evaluating approximate designs is to efficiently evaluate the output quality of approximate designs. Several *error metrics* have been used to evaluate the approximate designs. Error metric is a unit to measure the quality of the approximate designs compared to the correct designs. The most important error metrics are *Error rate* (ER) and *Error significance* (ES). ER represents the percentage of disparity between the correct and the approximate design. ES addresses the error magnitude. Other error metrics have been introduced, such as *Mean squared error* [7] to evaluate specific applications. Most designers evaluate their approximate designs using simulation, which is not accurate and time consuming. Chan et al.[8] proposed an automatic statical approach to estimate the output quality of the approximate designs which is able to evaluate the approximate designs using a number of error metrics. Although they proposed a regression-based technique to improve the accuracy of the EM estimation, EMs are still not absolute accurate. Additionally, the test patterns which produce the error output are important for implementing error detection and correlation logic, which is not addressed in [8].

In this paper, we propose an automatic formal approach of evaluating the imprecise adders based on binary decision diagrams (BDD) [9]. Our approach measures the imprecise adders using *Exact error rate* (EER), or exact error boundary, without any estimation. In addition, we take a fresh look at generating the test patterns which explicitly cover all the errors. This means that we are able to explicitly diagnose the errors which is important for improving the design qualities. We demonstrate our technique using number of Variable Latency Speculative Adder (VLSA) [2] and Accuracy Configurable Adder (ACA) [3] adders that are implemented using different size of group adders without error correlation logic. We make the following contributions:

1) We obtain the EER of a number for VLSAs and ACAs. We demonstrate that our approach is able to obtain the EER of 512-bit designs when group size $k = 32$ in 3 minutes.

2) We analyze the designs by measuring the EER of the design and the individual output bit. We show the EER distribution of the output bits in Section 5.

3) We introduce the modeling to obtain the EER of the design with ES range, e.g., the EER when the numerical difference between the correct and the actual results are in the range $[2^6, 2^8]$.

4) Our technique can explicitly generate the test patterns that cover all the errors. In addition, we show the distribution of these error input vectors (decimal).

II. Background

It is assumed that the reader is familiar with basic concepts of Boolean functions, Boolean networks, and BDDs. This section reviews basic terms related to approximate computing and formal analysis used throughout the paper.

A. Boolean function and BDDs

Bryant [9] introduced a concept of reduced, ordered BDDs (ROBDDs), along with a set of efficient operators for Boolean

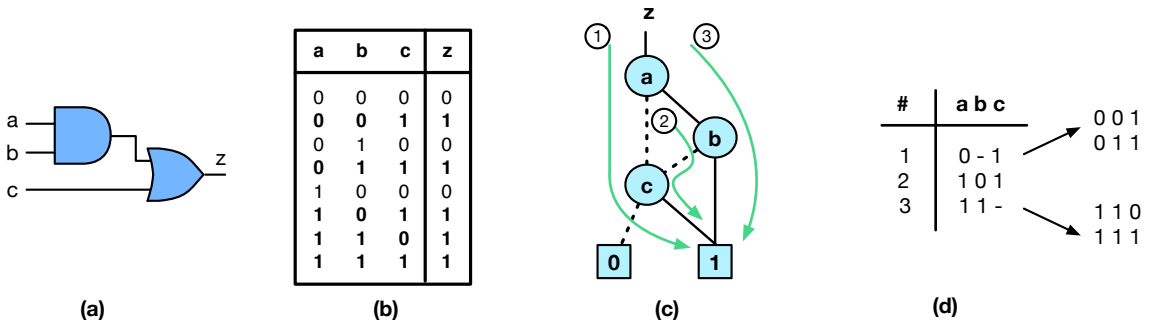


Fig. 1: *Boolean* function, *1-minterms*, and BDD.

function manipulation (symbolic manipulation), and proved the canonicity property of ROBDDs (Figure 2). Recently, a BDD-based minimization technique for approximate computing is presented in [1]. The authors proposed that the approximate circuit can be designed automatically by minimizing the BDD representation, which is limited to a error threshold. The error rate can be calculated in the minimization process. In this work, we study the quality of the imprecise adders by measuring the exact error rate using BDDs. Additionally, we propose the BDD-based test generation for imprecise designs.

A minterm is a product term with all variables for which the function evaluates to 1. Minterms which produce output in 1(0) are called *1-minterm* (*0-minterm*). It is well know that finding all the 1-minterms(0-minterms) can be done by searching all the paths from the root to node 1(0) in the BDD. Figure 1 (b) shows a truth table of the boolean logic in (a). This function has been converted into BDD, shown in Figure 1 (c). The 1-minterms can be obtained by the following paths: 1) $\bar{a} \rightarrow$c; 2) a $\rightarrow \bar{b} \rightarrow c$; 3) $a \rightarrow b$. The 1-minterms are shown in Figure 1 (d). We can see that these minterms may contain *don't cares* depending on the number of variables in each path. The number of input vectors covered by one minterm equals to 2^n, where n is the number of *don't cares* included in the minterm. Therefore, we calculate the probability of z=1 as (2+1+2)/2^3=5/8. Our approach of evaluating the EER and generating test patterns is modeled as collecting the 1-minterms using BDD, as presented in Section 3.1.

B. Imprecise Adders

Variable latency speculative adder (VLSA) is implemented by dividing the long addition into smaller groups of additions of length k [2]. The first group computes the lowest k-bit outputs (i.e. from LSB to (k - 1) bit). Other group adders are responsible for generating the output using the MSB of each group adder. For example, the carry-chain of a 16-bit VLSA when k=*8* (Figure 2) has been reduced from 16-bit to 8-bit addition. However, the area cost of this design is high since each group overlaps over 1 output bit.

To improve the area cost of the approximate adders, Error-Tolerant Adder (ETA) [4] has been introduced by dividing the long addition into a number of non-overlapping groups. However, this introduces many more errors. Accuracy configurable

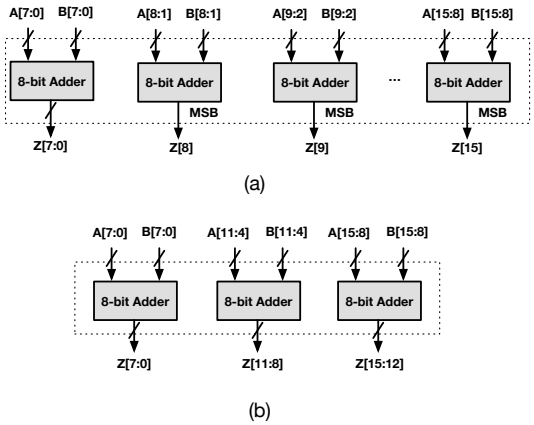


Fig. 2: (a) 16-bit VLSA, k=8 (b) 16-bit ACA, k=8

adder (ACA) [3] uses only the result of the upper half part of each addition so as to improve accuracy. A 16-bit ACA (k=8) is shown in Figure 2 (b). We can see that the carry-chain is reduced to 8-bit long addition and the number of group adders is reduced to 3.

C. Evaluation of Approximate Designs

The problem of evaluating imprecise adders can be informally defined as "measuring the dissimilarity between an imprecise adder and a regular adder". To quantify errors in approximate designs, error metrics have been introduced such as Error rate and Error significance. *Error rate (ER)* is the percentage of instances in which output value is different from the correct value. *Error significance (ES)* is the numerical difference between the correct and actual output results; this quantifies the amount of error. The designers evaluate the imprecise adders using random simulation methods [2][4][3]. Obviously, the simulation-based technique is limited by the runtime and evaluation accuracy. Error estimation techniques, such as interval-based approach [10], and statistical analysis [8], can quickly evaluate the approximate designs but suffer from estimation accuracy problems. For example, if the actual error distribution varies greatly within one interval, the estimation will be inaccurate. However, the absolute error evaluation has never been discussed. Additionally, test generation (TG) of approximate designs is important for diagnosis of errors

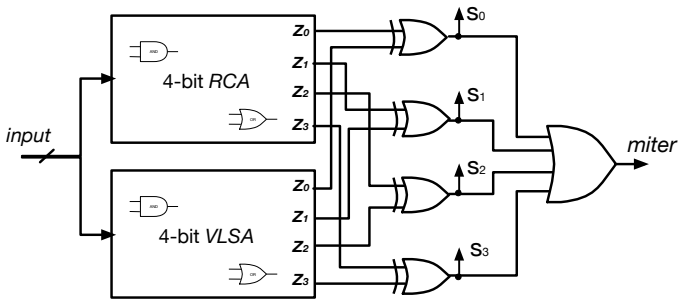


Fig. 3: Modeling

which can be used to efficiently design the correlation logic.

The basic notations are defined as following: assuming the imprecise adder is n-bit wide, $\mathcal{I}$=$\{i_1,\ i_2,\ ...,\ i_{2^{2n}}\}$ is the set of all input vectors; each i contains two operands. There is a subset of input vectors $\mathcal{I}_{error}$=$\{x_1,\ x_2,\ ...,\ x_m\}$ such that the output value of the imprecise adder is different compared to the correct adder (e.g. *RCA*). Exact error rate E = $m/2^{2n}$. Assuming that the function of the correct adder and the imprecise adder are $\mathcal{F}$ and $\mathcal{F}'$, ES can be represented as $\mathcal{D}$ = $\{\delta_1,\ \delta_2,\ ...,\ \delta_m\}$; where each δ_i = $\mid \mathcal{F}(x_i)$ - $\mathcal{F}'(x_i) \mid$.

III. Implementation

One of the key components of the hardware verification methodology is *Combinational Equivalence Checking* (CEC). Functional equivalence of two designs is formally solved using existing BDD or SAT based methods. Given two designs, the traditional *CEC* tool returns "equivalent" or "not equivalent". Equivalent means that for any input vector $\mathcal{I}$ = $\{i_1,i_2,...,\ i_n\}$ the two designs always produce the same output. If the designs are not equivalent, this means that there is a subset of input vectors $\mathcal{I}_{error}$ for which the two design produce different outputs. However, the current formal equivalence checking methods do not offer any method for finding $\mathcal{I}_{error}$.

A. Modeling

To complement our method with BDD, we formulate the problem using the modeling similar to *CEC*. The modeling is basically using XOR gates to check the equivalence, followed by a word wide OR gate, called "*miter*". Additionally, we add extra primary outputs which are the output signals of these XOR gates. Ripple-Carry Adder (*RCA*) is used as a reference design. The model of 4-bit VLSA (*k=2*) is shown in Figure 3. The extra primary outputs $\{s_0, s_1, s_2, s_3\}$ are used to evaluate the quality of the each output bit and *miter* is used for the entire design. Additionally, ES can be modeled using these extra POs as s_0+$2s_1$+$4s_2$+... +$2^{n-1}s_{n-1}$. The next step is to implement this model with BDD. If the two designs are equivalent, the BDD will be reduced to a *0-BDD*. Otherwise, the BDD is not empty and contains a number of paths from the root to node 1 in the BDD. These paths indicate the patterns which cause errors. We can collect all the input vectors for which the function evaluates to 1 by traversing the graph

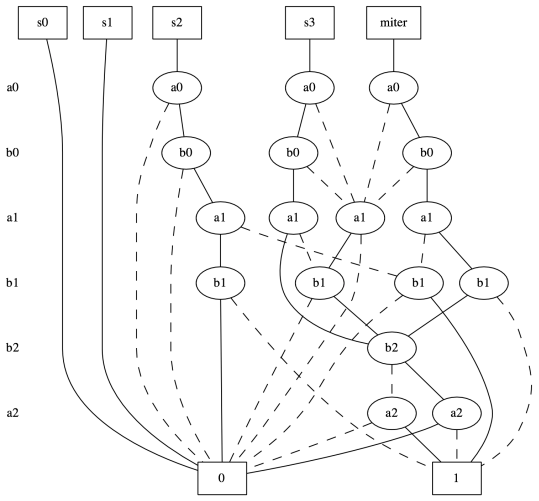


Fig. 4: *BDD* of 4-bit VLSA evaluation model

(Figure 1), and the EER can be calculated based on these patterns.

B. Case study: 4-bit VLSA

We illustrate this technique using a 4-bit VLSA (*k=2*), which contains three 2-bit adders. The first group adder produces output z_0, z_1 with two operands $A[1:0]$, and $B[1:0]$. The output bit z_2, z_3 are generated by two 2-bit adders (MSB) whose the operands are $A[2:1], B[2:1]$ and $A[3:2], B[3:2]$ respectively. The *BDD* of this model is shown in Figure 4. Based on the 4-bit *VLSA* design, we can see that output z_0 and z_1 are error-free. The BDD shows that s_0 and s_1 are *0-BDDs*, which proves that z_0 and z_1 are error-free. Meanwhile, we can see that s_2, s_3 and $miter$ are not *0-BDDs*, which means that they are not error-free. To measure the exact *Error rate*, we only consider the paths from the root to node 1. Each path represents a *1-minterm* in the model. The *Error rate* (ER) and test vectors can be automatically obtained by collecting all the paths to 1. For example, we can see that there are two paths from root s_2 to 1: ($a_0 \rightarrow b_0 \rightarrow a_1 \rightarrow \overline{b_1}$) and ($a_0 \rightarrow b_0 \rightarrow \overline{a_1} \rightarrow b_1$). The corresponding *1-minterms* or error test vectors of the 4-bit VLSA are shown in Table I ($\mathcal{I}_{error}$ of s_2 = $\{$(xx01,xx11), (xx11,xx01)$\}$). The total number of error input vectors is $4 \cdot 4 \cdot 2 = 32$. Hence, the ER of z_2 is $32/2^8 = 12.5\%$. We collect the error input vectors and ER of z_3 (18.75%) and the $miter$ (25%) using the same technique (Table I).

IV. Experimental Results

The technique described in this paper was implemented in C++ using CUDD 2.4.0 package [11]. The program was tested on a number of combinational imprecise adders [3][2] with different group sizes. The experiments were conducted on a PC with Intel Processor Core i5-3470 CPU 3.20GHz x4 with 15.6 GB memory. In this section, we demonstrate that our tool can efficiently evaluate the output quality of the imprecise adders using different parameters. Note that we are

TABLE II: Error rate of *VLSAs* and *ACAs* and the CPU runtime of evaluating 64-bit designs.

VLSA							*ACA*						
k	*16-bit*	*k*	*32-bit*	*k*	*64-bit*	*Runtime*	*k*	*16-bit*	*k*	*32-bit*	*k*	*64-bit*	*Runtime*
4	34.04%	8	4.636%	8	10.497%	0.11 s	4	47.79%	8	16.73%	8	34.85%	0.03 s
5	16.67%	10	1.702%	10	2.613%	0.24 s	6	20.98%	10	7.431%	10	15.65%	0.03 s
6	7.751%	12	0.244%	12	0.633%	0.26 s	8	5.859%	12	3.061%	12	6.757%	0.03 s
7	3.551%	14	0.0549%	14	0.152%	0.11 s	10	3.027%	14	1.160%	14	3.069%	0.03 s
8	1.563%	16	0.0112%	16	0.0366%	0.10 s	12	0.769%	16	0.389%	16	1.161%	0.03 s
9	0.684%	18	2.67E-3%	18	8.77E-3%	0.26 s	-	-	18	0.195%	18	0.584%	0.02 s
10	0.293%	20	5.72E-4%	20	2.10E-4%	0.10 s	-	-	20	9.76E-2%	20	0.244%	0.02 s

TABLE I: Bit-vector $\mathcal{I}_{error}$ of z_2, z_3 and the entire 4-bit *VLSA* (*x* = *don't care*)

s2		*s3*		*miter*	
A	B	A	B	A	B
xx01	xx11	x001	x111	x001	xx11
xx11	xx01	x010	x11x	x010	x11x
		x011	x101	x011	x001
		x011	x11x	x011	x101
		x110	x011	x011	x11x
		x111	x001	x101	xx11
		x111	x01x	x110	x01x
				x111	x001
				x111	x01x
				x111	x101

not evaluating the performance of the adders, i.e. the trade-off between accuracy and cost. We only measure the output quality of different imprecise adders.

A. Error Analysis

Table II shows the *Exact Error rate* (EER) of 16-bit, 32-bit and 64-bit *VLSAs* and *ACAs* with different group size *k*. The columns *Runtime* show the CPU time of evaluating the 64-bit designs. The EER of all the designs shown in Table II can be obtained in a fraction of a second. Additionally, we test our tool on a 512-bit *VLSA* adder with the group size of 32. It took only 152.2 seconds of CPU time and 28.2 MB memory.

Our tool also can evaluate the individual output bit quality using the extra primary outputs in the modeling presented in Section 3. We demonstrate this feature using the the same designs as in Table II. The results are shown in Figure 5. Figure 5(a), (b) and (c) show the error distribution of the output bits of 16-bit, 32-bit, and 64-bit *ACA* adders. Figure 5 (d), (e) and (f) show the error distributions of the output bits of 16-bit, 32-bit, and 64-bit *VLSA* adders. The CPU runtime of this experiment is included in Table II. We can see that the error rate of the output bits, which are generated by each group adder in the *ACA* adder (except the first one), are the same. This is because the length differences between the actual carry chain and the correct carry chain of all group adder in *ACA*, are the same. Regarding the *VLSA* adders, we can see that the error rate of the output bits increases to a constant after a certain bit position p. Additionally, we observe that p can be represented as a function of *k*, i.e. the size of the group adder.

In this paper, we also offer an advanced model to evaluate the imprecise adders with a specific error range. The basic idea is to insert extra logic for defining the error range

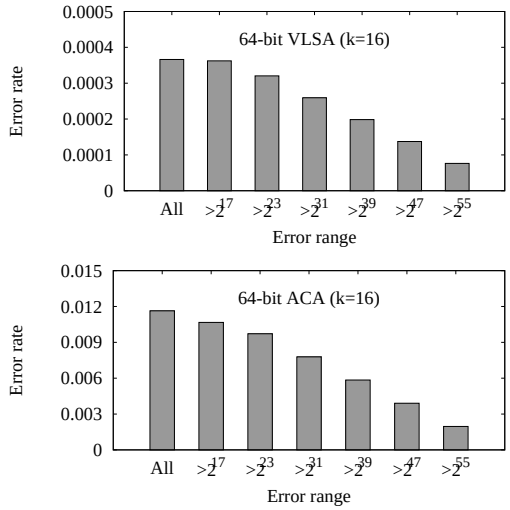
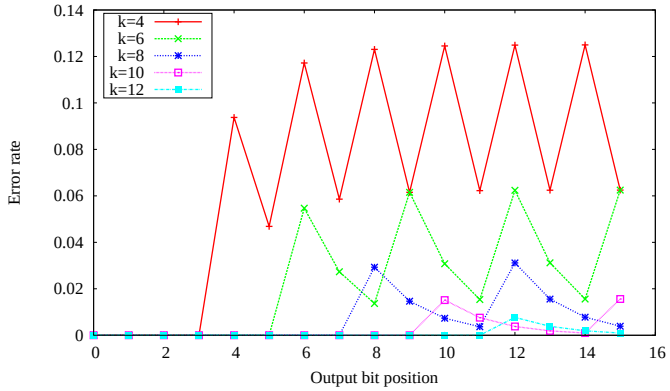


Fig. 6: Error rate with error significance range.

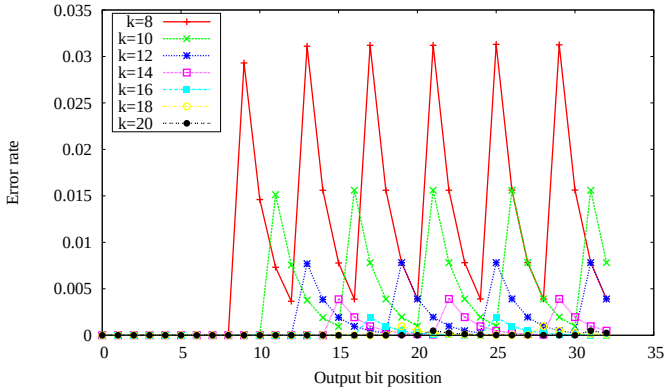
constrain. The *Error significance* (ES) is the numerical difference between correct and output results, which can be represented using signals s_i in our model, i.e. $\mathcal{S}$ = $2^0 s_0$+$2^1 s_1$ ... +$2^{n-1} s_{n-1}$. In this paper, we add extra comparators ("¡", smaller than) to model the error range. $\mathcal{S}$ is one of the input of the comparators and the other input is a constant number based on the error range. For example, assuming there is a 8-bit VLSA and the given error range is [32,64], two 8-bit comparators are required. In the first comparator the input operands are $\mathcal{S}$ and *00100000* and the output is c_1. In the second comparator the input operands are $\mathcal{S}$ and *01000000* and the output is c_2. Then, we create a new "miter" m_{new} = AND($\bar{c_1}$, c_2). This error rate equals the probability that m_{new} evaluates to 1. We tested this method using 64-bit VLSA and ACA with the group adder of size 16-bit (Figure 6). The runtime with or without error range is the same. This is because the extra logic does not introduce new variables.

B. Test generation (TG)

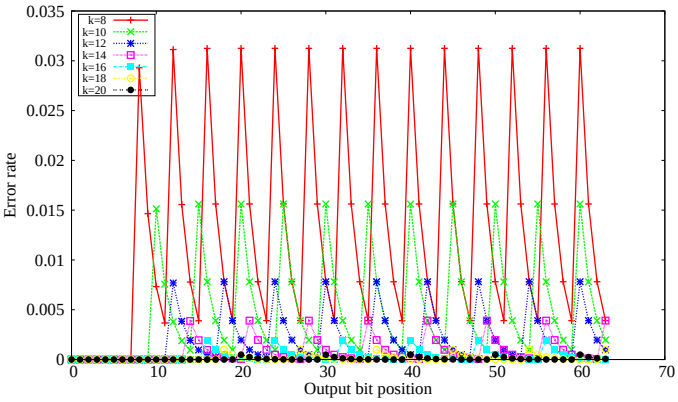
The method to explicitly generate the test vectors that cover the error of the imprecise adders is shown in Section 3.2 (Table 1). The result of TG using our technique is shown in Table III. The 16-bit VLSA(ACA) has more than 1460(2000) million error input vectors for $k = 4$. We can see that our tool can generate the test patterns which cover all the error inputs in 5 seconds. Additionally, we observe that the runtime of TG does not increase as the number of errors increasing. The runtime of



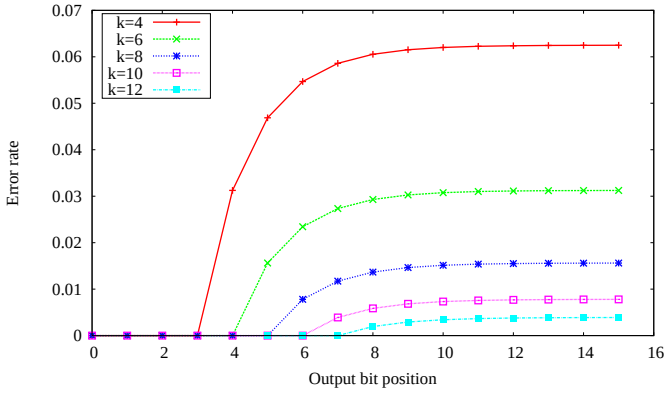
a Error distribution of 16-bit *ACA* (k=4,6,8,10,12)



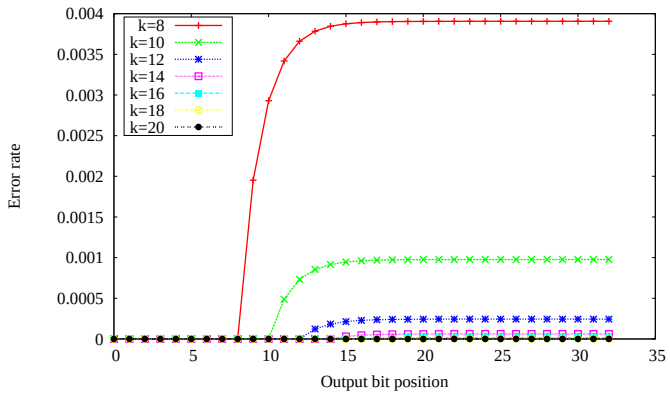
b Error distribution of 32-bit *ACA* (k=8,10,12,14,...,20)



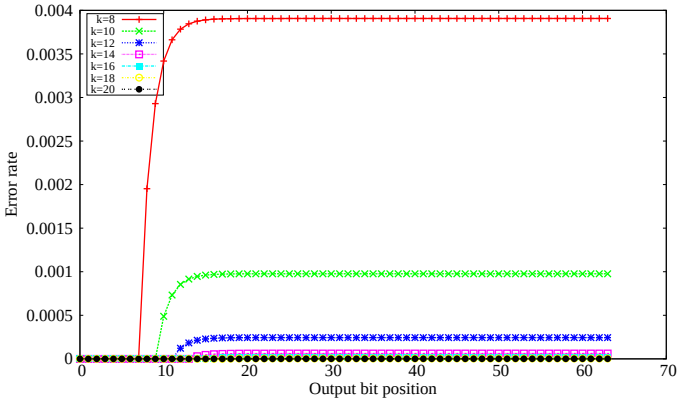
c Error distribution of 64-bit *ACA* (k=8,10,12,14,...,20)



d Error distribution of 16-bit *VLSA* (k=4,6,8,10,12)



e Error distribution of 32-bit *VLSA* (k=8,10,12,14,...,20)



f Error distribution of 64-bit *VLSA* (k=8,10,12,14,...,20)

Fig. 5: Output-bit error distribution of 16-bit, 32-bit, and 64-bit VLSA and ACA.

TG depends how many test patterns exist, that is the number of paths from the root the 1 in the BDD. For example, a single test pattern (e.g. (1xxx)) may cover 8 errors; two test patterns (e.g. (1x01), (x011)) may cover 4 errors.

In addition to TG, we expand the test vector without the *don't cares* to see the distribution of the error input vectors in decimal. The results are shown in Figure 8. $Op1$ and $Op2$ represent the decimal value of the two operands of the adders. Each dot in the figure represents one input. We observe that: **1)** errors happen when the correct addition ($Op1+Op2$) equals to some certain values (e.g. when the correct addition equal to "128" in Figure 7); **2)** the densities of the distributions clearly indicate the structural difference between *VLSA* and *ACA* designs. The length of the missing carry chain in all group adders of ACA are 4. However, for $VLSA$, the length is increasing until the $(k-1)^{th}$ group adder.

C. Analysis of Imprecise Multipliers

We also applied our technique to evaluate the 8-bit imprecise multipliers. A typical multiplier consists of three stages: partial product generation, partial product accumulation and a final stage adder. One simple idea of building an imprecise

TABLE III: Runtime of Test generation. # TP = The number of test patterns.

16-bit VLSA			*16-bit ACA*		
k	# TP	*Runtime*	*k*	# TP	*Runtime*
12	100,352	0.20 s	12	23,294	0.23 s
10	319,488	0.36 s	10	1,293,266	1.16 s
8	1,039,873	0.83 s	8	68,478	0.89 s
6	3,174,688	2.18 s	6	3,721,846	3.04 s
4	11,025,319	4.63 s	4	1,444,494	2.36 s

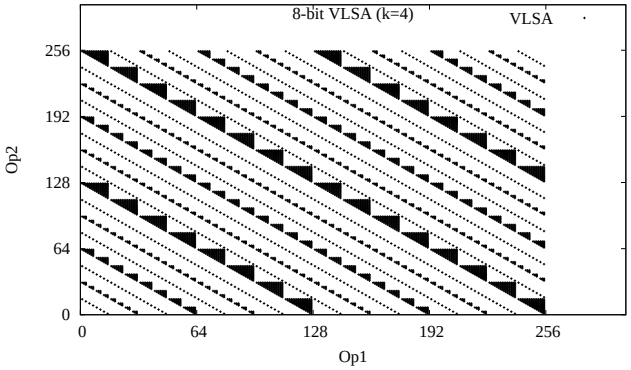


Fig. 7: Error inputs (decimal) distribution of 8-bit *VLSA*; *Op1* and *Op2* are two operands of the adder.

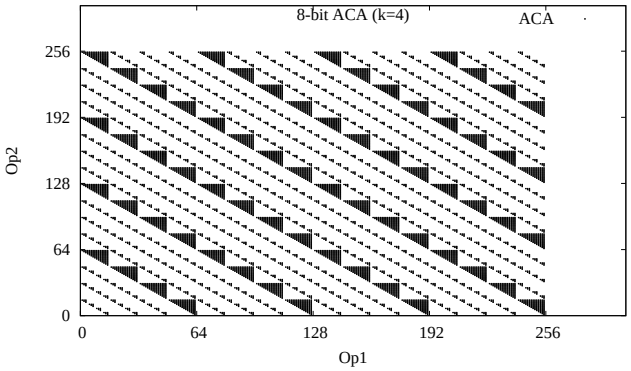


Fig. 8: Error inputs (decimal) distribution of 8-bit *ACA*; *Op1* and *Op2* are two operands of the adder.

multiplier is replacing the final stage adder with an imprecise adder [10]. We implemented the imprecise multipliers by replacing the last one stage adder or last two stage adders. As shown in Figure 9, k is the group adder size of the imprecise adder and s is the number of stages replaced by imprecise adders. The imprecise adder used here is VLSA. There are three observations. First, the error of each output bit increases when the number of stages replaced by imprecise adders increases. When $s = 2$, the error of each output bit is almost 1x greater than $s = 1$. Meanwhile, this introduces one more error output bit. Second, the number of errors increases when k decreases. Third, compared to the result of VLSA in Figure 5, we can see that the distribution of the imprecise adder is changed. The reason is that the set of input vectors of the VLSA in the imprecise multiplier are different than evaluating the VLSA separately. In the imprecise multiplier, the input vectors of the imprecise adder are reduced by the partial product generation and accumulation logic.

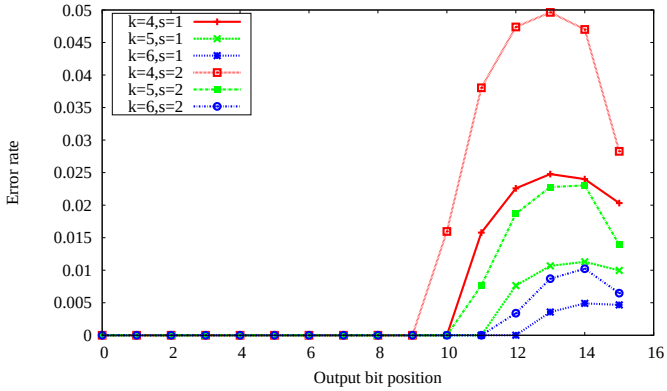


Fig. 9: Error distribution of 8-bit imprecise multiplier.

V. Conclusion

We proposed an approach that is able to evaluate the output quality of imprecise adders with absolute accurate measurements. Our tool can efficiently measure the exact error rate of the design and the individual output bits. This method is able to generate the test patterns which cover all the errors produced by the imprecise adders, with or without a certain error range. Our test generation (TG) is able to generate more than 10 million test patterns which cover 1.5 billion errors in 5 seconds. We also evaluated the 8-bit imprecise multipliers. Our future work will focus on evaluating large imprecise multipliers.

References

[1] M. Soeken, D. Große, A. Chandrasekharan, and R. Drechsler, "BDD Minimization for Approximate Computing," in *ASP-DAC'16*. IEEE, Jan. 25, 2016, pp. 474–479.

[2] A. K. Verma, P. Brisk, and P. Ienne, "Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design," in *DATE*. ACM, 2008, pp. 1250–1255.

[3] A. B. Kahng and S. Kang, "Accuracy-configurable Adder for Approximate Arithmetic Designs," in *49th DAC*. ACM, 2012, pp. 820–825.

[4] N. Zhu, W. L. Goh, and K. S. Yeo, "An Enhanced Low-power High-speed Adder for Error-tolerant Application," in *Integrated Circuits, ISIC'09. Proceedings of the 2009 12th International Symposium on*. IEEE, 2009, pp. 69–72.

[5] N. Zhu, W. L. Goh, W. Zhang, K. S. Yeo, and Z. H. Kong, "Design of Low-power High-speed Truncation-error-tolerant Adder and its application in Digital Signal Processing," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 8, pp. 1225–1229, 2010.

[6] G. Liu, Y. Tao, M. Tan, and Z. Zhang, "CASA: Correlation-aware Speculative Adders," in *Proceedings of the 2014 international symposium on Low power electronics and design*. ACM, 2014, pp. 189–194.

[7] Z. Wang and A. C. Bovik, "Mean Squared Error: Love it or Leave it? a new Look at Signal Fidelity Measures," *Signal Processing Magazine, IEEE*, vol. 26, no. 1, pp. 98–117, 2009.

[8] W.-T. J. Chan, A. Kahng, S. Kang, R. Kumar, and J. Sartori, "Statistical Analysis and Modeling for Error Composition in Approximate Computation Circuits," in *Computer Design (ICCD), 2013 IEEE 31st International Conference on*. IEEE, 2013, pp. 47–53.

[9] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 677–691, 1986.

[10] J. Huang, J. Lach, and G. Robins, "A Methodology for Energy-quality Tradeoff Using Imprecise Hardware," in *49th DAC*. ACM, 2012, pp. 504–509.

[11] F. Somenzi, "CUDD: CU Decision Diagram Package-release 2.4. 0," *University of Colorado at Boulder*, 2009.