Efficient Batch Statistical Error Estimation for Iterative Multi-level Approximate Logic Synthesis

Sanbao Su, Yi Wu, Weikang Qian

University of Michigan-Shanghai Jiao Tong University Joint Institute
Shanghai Jiao Tong University, Shanghai, China
Email: {gawaine, eejessie, qianwk}@sjtu.edu.cn

ABSTRACT

Approximate computing is an emerging energy-efficient paradigm for error-resilient applications. Approximate logic synthesis (ALS) is an important field of it. To improve the existing ALS flows, one key issue is to derive a more accurate and efficient batch error estimation technique for all approximate transformations under consideration. In this work, we propose a novel batch error estimation method based on Monte Carlo simulation and local change propagation. It is generally applicable to any statistical error measurement such as error rate and average error magnitude. We applied the technique to an existing state-of-the-art ALS approach and demonstrated its effectiveness in deriving better approximate circuits.

CCS CONCEPTS

• **Hardware → Combinational synthesis**;

KEYWORDS

approximate logic synthesis, approximate computing, logic synthesis, error estimation

1 INTRODUCTION

Energy efficiency has become a major concern for designing VLSI circuits as transistor size goes into nano-scale [1]. Meanwhile, many applications used today, such as machine learning, artificial intelligence, and multimedia, are inherently error-resilient. Given these trends, approximate computing [2] was proposed as a novel energy-efficient design paradigm for these error-resilient applications. It trades off accuracy for area, delay, and power consumption of circuits. Many research works [3] have demonstrated its capability to significantly improve the final circuit quality under a small amount of computation error.

There are two main research fields in approximate circuit design: manual design and automatic synthesis. The former manually designs approximate circuits such as adders [4] and multipliers [5]. The latter designs algorithms to synthesize a good approximate version for an arbitrarily given circuit. It involves approximate high-level synthesis [6, 7] and approximate logic synthesis (ALS). In ALS, many studies focused on synthesizing multi-level circuits [8–17], due to their wide use.

Given the large design space of an approximate circuit, many multi-level ALS methods [8–14] are based on iteratively applying the best local *approximate transformations (ATs)*. We call them *greedy iterative ALS flow*. In each iteration of such a flow, all valid local ATs are identified. Then, the quality (such as area, delay, or power consumption) improvement and the induced error, such as error rate (ER) or average error magnitude (AEM), of each AT are evaluated. The one with the largest score, which is typically calculated as the ratio of the quality improvement over the error, is then selected and applied. In this case, an accurate error estimation is important. If the error is not estimated very accurately, the greedy approach may choose a worse local transformation, which could lead to a worse final approximate design. Furthermore, a wrong error estimation may reduce the number of iterations and hence, the final quality, because it may select transformations with larger errors and consume the allowed error margin more quickly.

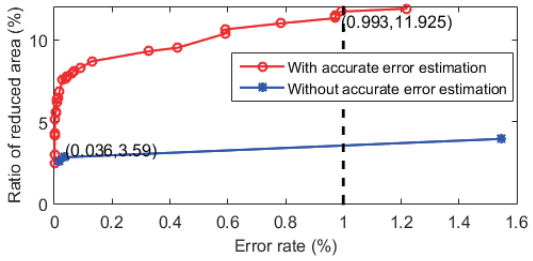

Figure 1: Error rate versus ratio of area reduction for the same ALS flow SASIMI with and without accurate error estimation. The benchmark is c7552 and the error rate threshold is 1%.

A motivating example is shown in Fig. 1. It shows the effect of the accurate error estimation to an existing ALS flow SASIMI [10]. The red curve and the blue curve show how the SASIMI methods with and without accurate error estimation, respectively, work on the benchmark circuit c7552. The error constraint is that the ER should be no more than 1%. Each point on a curve corresponds to the result after one iteration. From the blue curve, we can see that the method without accurate error estimation stops after 3 iterations. This is because at the third iteration, due to the wrong error estimation, the best candidate AT it selects actually increases the ER by 1.5%, which causes the flow to reach the error limit. In contrast, with accurate error estimation, candidate ATs with smaller induced error can be found. This leads to more iterations and better quality for the final approximate circuit, as shown by the red curve. For this example, the SASIMI method with accurate error estimation can reduce 8.3% more area than that without accurate error estimation.

From the above example, we can see that an accurate error estimation is helpful to a greedy iterative ALS flow. However, on the other hand, an accurate *batch error estimation* for all candidate ATs is time-consuming, because in each iteration, the approximate circuit with each possible AT applied should be simulated to obtain the accurate error. Thus, to improve both the quality and runtime of a greedy iterative ALS flow, an efficient and accurate batch error estimation method for all candidate ATs is in demand.

In this paper, we propose a batch error estimation method based on Monte Carlo simulation and local change propagation to improve the estimation accuracy with small runtime overhead. This approach can be applied to any graph-based representation of circuits, such as AND-inverter graph (AIG) [18], majority-inverter graph (MIG) [19], and gate netlist after technology mapping. The approach is also applicable to any statistical error measure, such as ER and AEM. To make it generally applicable to any input distribution, our method is based on Monte Carlo (MC) simulation. To

DAC '18, June 24–29, 2018, San Francisco, CA, USA

ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3196038

avoid the time-consuming MC simulation for all the ATs, we develop an efficient local change propagation method that can capture the effect of an internal approximation-induced error on all the primary outputs. We demonstrate that the time complexity of our method is much smaller than that of the full simulation method to obtain errors. We apply our proposed technique to a state-of-the-art ALS flow, SASIMI [10]. Our experiments showed that our method improved the circuit quality for both the ER and the AEM constraints compared to the original SASIMI flow.

The rest of this paper is organized as follows. Section 2 discusses the related works. Section 3 presents the background. Section 4 introduces our error estimation method in detail. Section 5 shows the experimental results. Finally, Section 6 concludes the paper.

2 RELATED WORK

There exist a number of works on ALS for multi-level circuits. We briefly discuss them in this section and point out whether our proposed batch error estimation technique is suitable for them or not. The work [9] proposes a systematic method that encodes the error constraint through a circuit and utilizes external don't cares for simplification. Nevertheless, its error constraint is maximum error magnitude, which is not a statistical error measure. Thus, our method is not applicable. The work [17] proposes a statistically certified approach with stochastic optimization under different error constraints. In each iteration, it randomly selects an AT and accepts it probabilistically. For early iterations, our proposed method may not be helpful because there is only one AT under consideration and hence, a direct error evaluation is affordable. However, we may apply our batch technique in later iterations when the accumulated error is close to the limit: in this case, due to the reduced error margin, it may be advantageous to consider multiple candidates and then choose a good one.

Several other works are greedy iterative ALS flows for statistical error measure. Thus, our method can help improve the accuracy of error estimation. In [8], Shin and Gupta proposed to inject stuck-at-faults to simplify circuits under ER and error magnitude constraint. Its AT is to set a signal to a constant 0 or 1. In [10], Venkataramani *et al.* presented an approach called SASIMI, which is able to handle either ER or AEM constraint. Its basic AT is to identify a pair of almost identical signals and substitute one with the other in the pair. In [11], Wu and Qian proposed a novel approach for combinational circuits under ER constraint. Its basic AT is to shrink a node in a Boolean network by deleting some literals from the node's factored-form expression. The work [13] proposes an ALS approach for FPGA designs. Its basic AT is to remove one input of a local circuit and then reconfigure the local function.

3 BACKGROUND

3.1 Statistical Error Measure

In this work, we consider error specification as a statistical error measure. Two typical statistical error measures are *error rate (ER)* and *average error magnitude (AEM)*. ER is the total probability of all the input combinations that produce wrong outputs. AEM is the mean numerical deviation of the binary number encoded by the approximate outputs. AEM is usually used for arithmetical circuits.

3.2 Greedy Iterative ALS Flow

As we pointed out in Section 2, many existing multi-level ALS methods belong to the greedy iterative flow. The basic idea of this flow is to gradually improve the circuit by applying a local AT in each iteration. An AT is a small perturbation to the netlist. Several examples of AT can be found in Section 2. In each iteration, the flow chooses the local optimal AT and applies it to the current approximate circuit C_{App}. For this purpose, all possible ATs of C_{App} are first gathered. Then, the improved quality IQ and the increased error IE for each AT are evaluated, where the quality could be area, delay, or power consumption and the error could be ER or AEM. Note that both the improved quality and the increased error for an AT are calculated over the current approximate circuit. Then, the AT that maximizes a score function over IQ and IE is selected and applied. A typical score function is IQ/IE, which favors an AT that maximizes the improved quality while minimizing the increased error. The final step of each iteration is to calculate the actual error of the new approximate circuit. If the actual error is smaller than the given error threshold, then the next iteration begins; otherwise, the flow finishes and the latest approximate circuit is returned.

4 METHODOLOGY

As we mentioned in Section 1, an efficient and accurate batch error estimation method is important to ALS. If we can estimate error accurately, we can choose the local optimal approximate transformation in each iteration. We can also avoid premature termination of the ALS flow. In this section, we describe our method for an efficient and accurate batch error estimation.

4.1 Monte-Carlo Based Error Estimation

Our proposed approach is based on Monte Carlo (MC) simulation. We first argue the necessity of using MC simulation in estimating the error. For a statistical error such as ER and AEM, which is of interest in this work, there are usually two ways to obtain it: analytical methods, which are based on signal probability propagation [20] or binary decision diagram (BDD) [21], and MC simulation. The analytical methods only work when all the input bits are independent, which may not be true for a general input distribution. Thus, to make the approach more general, MC-based logic simulation should be applied. Strictly speaking, a MC simulation cannot give the exact result due to random variation. However, by the law of large numbers, if a sufficient number of samples are used, the final result will be very close to the exact value [22].

An important step in the greedy iterative ALS flow is to do batch evaluation of the increased errors for all candidate ATs in one iteration. To obtain the accurate error for *each* AT, the circuit $C_{App,AT}$ obtained by applying that AT to the current approximate circuit C_{App} should be simulated and the simulation result should be compared with that of C_{App}. Thus, to get the accurate errors for all candidate ATs, the total number of MC runs is equal to the number of ATs, which significantly increases the runtime.

To improve the runtime efficiency, some previous methods just use the error observed at the output of the local circuit affected by the AT as an estimate to the final exact error [11, 13]. In this case, since we do not need to propagate the logic simulation values from the output of the local circuit affected by each AT to the primary outputs of the circuit, only one MC run is required to obtain the errors for all ATs. However, since this approximation ignores the potential logic masking effect from the output of a local circuit to the primary outputs, the error estimation could be quite inaccurate.

In this work, we propose a method to enhance the accuracy of batch error estimation by still using a single MC run. It involves two steps: 1) obtaining a change propagation matrix and 2) batch calculating the increased errors for all candidate ATs. We describe these two steps in detail in the next two subsections.

4.2 Obtaining the Change Propagation Matrix

The batch error estimation is based on a single MC run. Assume that we have M random input patterns, which are sampled from a given input distribution. For each input pattern, we apply logic simulation and obtain the logic values for all the nodes in the netlist.

Assume the netlist contains N nodes, among which there are O primary output nodes. In order to obtain the increased error for each AT, we need to determine whether a value change occurred at the output of the local circuit affected by the AT will be propagated to each primary output. To capture the local change propagation, we define a three-dimensional 0-1 change propagation matrix (CPM) P of size $M \times N \times O$. Each entry in the CPM is indexed as $P[i, n, o]$,

where $1 \le i \le M$ indicates the i-th input pattern in the MC run, n represents a node in the netlist, and o represents an output of the netlist. If the entry $P[i, n, o] = 1$, it means a change on the node n can be propagated to the output o under the i-th input pattern; otherwise, it cannot. The CPM is obtained based on Boolean difference, defined as follows.

Definition 4.1. Given a function f, its *Boolean difference (BD)* with respect to a variable x is denoted as $\frac{\partial f}{\partial x}$ and computed as:

$$\frac{\partial f}{\partial x} = f_x \oplus f_{\bar{x}}, \tag{1}$$

where f_x and $f_{\bar{x}}$ are the *positive cofactor* and the *negative cofactor* of f with respect to x, respectively. f_x and $f_{\bar{x}}$ are obtained by setting x to 1 and 0 in f, respectively. □

The BD $\frac{\partial f}{\partial x}$ is a function on all the other input variables of f except x. By its definition, we can see that if a combination of all the other input variables of f except x lets the BD $\frac{\partial f}{\partial x}$ be 1, then for this combination, the value of f will change when x changes.

For each node n in the netlist, we define S_n as the set of fanouts of the node n. For each $1 \le i \le M$, each node n in the netlist, and each node $n_f \in S_n$, we use the variable $D[i, n, n_f]$ to represent the value of BD $\frac{\partial n_f}{\partial n}$ under the i-th input pattern. To efficiently obtain the CPM, we need to obtain the value for each $D[i, n, n_f]$. It can be obtained by two steps. First, we compute the BD $\frac{\partial n_f}{\partial n}$ by Eq. (1). This gives a function g on all the other inputs of n_f except n. Then, we apply the values of the other inputs under the i-th input pattern to the function g. This gives the value $D[i, n, n_f]$.

Example 4.2. In Fig. 2a, suppose the function of N_1 is $I_1 I_2$. By Eq. (1), the BD $\frac{\partial N_1}{\partial I_1} = f_{I_1} \oplus f_{\overline{I_1}} = I_2 \oplus 0 = I_2$. If for the i-th input pattern, the value of I_2 is 1, then $D[i, I_1, N_1] = 1$. □

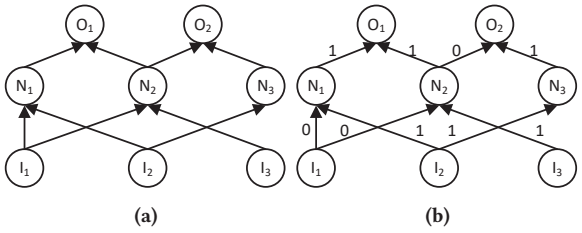

Figure 2: (a) A netlist of abstract nodes; (b) The node netlist with values of Boolean differences.

If $D[i, n, n_f] = 1$, then when we apply an AT to the node n and the value of n is changed under the i-th input pattern, the value of n_f will also be changed.

Now, we show how we obtain the CPM. For each fixed i, the values of $P[i, n, o]$ for all nodes n and outputs o can be obtained by a reverse topological traverse over the netlist from the primary outputs to the primary inputs. We start from the output nodes. For each output node o, since a change on it can be observed at the node itself, we have $P[i, o, o] = 1$. For any other node n, the value $P[i, n, o]$ can be recursively calculated. Indeed, a change on n can be observed at the output o when there exists a fanout of n, n_f, such that the change on n causes a change on n_f and the change on n_f can be observed at the output o. Note that if a change on n causes a change on n_f, we must have $D[i, n, n_f] = 1$. If a change on n_f can be observed at the output o, we must have $P[i, n_f, o] = 1$. Thus, we can conclude the following recursive formula for calculating $P[i, n, o]$:

$$P[i, n, o] = \bigvee_{\forall n_f \in S_n} (P[i, n_f, o] \wedge D[i, n, n_f]), \tag{2}$$

where S_n is the set of fanouts of node n.

Example 4.3. Consider the netlist in Fig. 2b and the i-th simulation slice. For each node n in the netlist and each fanout n_f of node n, the value of $D[i, n, n_f]$ is shown near the directed edge $e(n, n_f)$. For example, the value of $D[i, N_1, O_1]$ is 1.

Since O_1 and O_2 are the outputs, we have $P[i, O_1, O_1] = P[i, O_2, O_2] = 1$. By applying Eq. (2) consecutively, we can obtain

$$\begin{aligned} P[i, N_1, O_1] &= P[i, O_1, O_1] \wedge D[i, N_1, O_1] = 1, \\ P[i, N_2, O_1] &= P[i, O_1, O_1] \wedge D[i, N_2, O_1] = 1, \\ P[i, I_1, O_1] &= (P[i, N_1, O_1] \wedge D[i, I_1, N_1]) \vee \\ &\quad (P[i, N_2, O_1] \wedge D[i, I_1, N_2]) = 0. \end{aligned}$$

Thus, a change on I_1 cannot be propagated to the output O_1 under this input pattern. □

4.3 Batch Error Calculation

After we obtain the CPM, we can use it to do batch error calculation. We first discuss how we handle error specification as ER. The batch ER calculation is simply calculating the increased ER of each AT individually, but using the same CPM. The flow for obtaining the ER for an AT is shown in Algorithm 1. The algorithm takes the previous approximate circuit C, the CPM P, and an approximate transformation under consideration as inputs. It first identifies the output n_x of the local circuit in C affected by the AT (see Line 4). Then, we go through all the M simulation slices and accumulate the increased ER. For each simulation slice i, we need to judge whether the value of n_x changes after applying this AT (see Line 6). This can be achieved by considering the inputs to the local circuit and obtain the output of the local circuit after applying AT. If the value of n_x does not change, then for the current simulation slice, all primary output values do not change after applying the AT. Consequently, the increased ER is 0. Otherwise, the change at the local output n_x may be propagated to some primary outputs and this may induce a non-zero increased ER. Next, we will discuss how we obtain the increased ER if the output n_x of the local circuit changes after applying the AT for an input pattern. This corresponds to Lines 6–11 in Algorithm 1.

Algorithm 1: Algorithm for estimating the increased ER for an approximate transformation.

```
1  Input: Netlist C of the previous approximate circuit, change
   propagation matrix P, and an approximate transformation AT;
2  Output: Increased error rate ER;
3  Increased error rate ER ← 0; M ← number of input patterns;
4  Identify the output n_x of the local circuit affected by AT;
5  for i from 1 to M do
6      if the values at n_x before and after applying AT are different then
7          if there exist some wrong outputs in the previous approximate C
             then
8              if for each wrong output o_w, P[i, n_x, o_w] = 1 and for
                 each right output o_r, P[i, n_x, o_r] = 0 then
9                  ER ← ER − 1/M;
10         else if there exists an output o such that P[i, n_x, o] = 1 then
11             ER ← ER + 1/M;
12 return ER
```

One thing to note is that the increased ER is measured against the approximate circuit obtained in the previous round, i.e., the input netlist C. In order to obtain the increased ER more accurately, in our approach, we use the same set of random input patterns for all the iterations in the ALS flow. Thus, the same i-th input pattern is applied to the approximate circuits before and after applying the AT. Given this, by comparing the outputs of the two circuits, we are able to estimate the increased ER.

Comparing the approximate circuits before and after applying the AT, there are two situations in which the ER change is non-zero:

(1) Before applying the AT, the approximate circuit C has some wrong outputs and after applying the AT, the new circuit has all the outputs correct. In this case, the ER decreases. Since we are considering 1 simulation slice out of M, the change is $-1/M$ (see Line 9). This situation occurs when there exist some wrong outputs in the previous approximate circuit C (see Line 7). Furthermore, the change at the node n_x can be propagated to all those previously wrong outputs, which means the change now makes all these outputs correct. Meanwhile, for all those previously correct outputs, the change at the node n_x cannot be propagated to them. Thus, after applying the AT, all the outputs are correct. These conditions are summarized in the **if** statement at Line 8.

(2) Before applying the AT, the approximate circuit C has all the outputs correct and after applying the AT, the new circuit has some wrong outputs. In this case, the ER increases by $1/M$ (see Line 11). This situation occurs when there is no wrong output in the previous approximate circuit C. Furthermore, the change at the node n_x can be propagated to some outputs. The latter condition corresponds to the condition in the **else if** statement at Line 10.

Note that to efficiently track the set of wrong outputs for each input pattern for an approximate circuit, we maintain an $M \times O$ 0-1 matrix W. For each $1 \leq i \leq M$ and each output o, the entry $W[i, o]$ is 1 if and only if the output o is wrong for the i-th input pattern. This matrix is updated in each iteration of the ALS flow.

Finally, after we have iterated over all the input patterns, we obtain the increased ER and the algorithm finishes.

If the error specification is AEM, we can obtain the increased AEM of an AT in a similar way. To keep track of the increased AEM over the previous approximate circuit, we maintain two $M \times O$ 0-1 matrices V and U. For each $1 \leq i \leq M$ and each output o, the entries $V[i, o]$ and $U[i, o]$ record the values at the output o of the previous approximate circuit and the original circuit, respectively, for the i-th input pattern. Assume the binary numbers encoded by the outputs of the original circuit, the previous approximate circuit, and the approximate circuit after applying the AT for the i-th simulation slice are $Y_{org}[i]$, $Y_{pre}[i]$, and $Y_{chg}[i]$, respectively. Then, the increased AEM can be calculated as

$$AEM = \sum_{i=1}^{M} \left(|Y_{chg}[i] - Y_{org}[i]| - |Y_{pre}[i] - Y_{org}[i]| \right).$$

The numbers $Y_{org}[i]$ and $Y_{pre}[i]$ can be directly obtained from the matrices U and V, while the number $Y_{chg}[i]$ can be efficiently obtained through the CPM. Specifically, from the CPM, we can efficiently obtain the set of changed outputs for an AT under consideration. Then, by flipping the corresponding entries in the matrix V, we can obtain $Y_{chg}[i]$.

Since the increased error obtained by our method is over the previous approximate circuit, the amount of increased error may even be negative for some ATs, which are favorable choices. Due to the high accuracy of our proposed error estimation algorithm, we are able to identify these favorable ATs and improve the quality of the synthesized approximate circuit.

Finally, we want to point out that sometimes, our method may still be incorrect. One reason for this is the existence of reconvergent paths. For example, in Fig. 2a, I_1 can reach O_1 through either the path $I_1 \rightarrow N_1 \rightarrow O_1$ or the path $I_1 \rightarrow N_2 \rightarrow O_1$. Assume that for an AT and the i-th input pattern, the value of I_1 changes and it causes further changes to N_1 and N_2. Note that the value $D[i, N_1, O_1]$ is a function on N_2. In our approach, we still use the original value of N_2 to compute $D[i, N_1, O_1]$. However since the value of N_2 changes, such a computation is wrong and consequently, the value of $P[i, I_1, O_1]$, which indirectly depends on $D[i, N_1, O_1]$ as shown in Example 4.3, may also be wrong. This influences the accuracy of our error estimation method. Solving such a problem is one of our future tasks. Despite this problem, we still empirically observe that in most cases, our approach gives a highly accurate result.

4.4 Time Complexity Analysis

In this section, we analyze the time complexity of our proposed algorithm for batch error estimation of all candidate ATs in an iteration of the ALS flow. We consider the time complexity for estimating the ER; that for estimating AEM is the same. Assume the number of candidate ATs is T and the number of input patterns for simulation is M. Assume the circuit contains N nodes, E edges, and O outputs. Our proposed method involves two major steps: 1) obtaining the CPM and 2) calculating the ERs for all the candidate ATs. The time complexity of obtaining the CPM is $\Theta(M(N+E)O)$, where M is due to the loop over all M input patterns, O is due to the loop over all O outputs, and the term $(N+E)$ is due to the reverse propagation of the $P[i, n, o]$ values. To calculate the ERs for all the candidate ATs, we need to apply Algorithm 1 T times. Since the time complexity of Algorithm 1 is $\Theta(MO)$, the total time complexity to obtain the ERs for all the candidate ATs is $\Theta(MTO)$. Thus, the time complexity of our method is $\Theta(MO(N+E+T))$. For a general circuit, which can be modelled as a sparse graph, its number of edges, E, is on the same order of its number of nodes, N. For a typical ALS method, the number of candidate ATs is at least of the same order of N. Thus, $N+E+T = \Theta(T)$ and the time complexity of our method is $\Theta(MOT)$.

For comparison purpose, consider the method to run the MC simulation for each candidate AT to obtain its accurate increased ER. In order to get the ER for one AT, the simulation runtime is $\Theta(M(N+E))$, where M is due to the M simulation slices and $(N+E)$ is due to the forward propagation of the input values to the outputs. Given that there are T ATs in total, the time complexity of the full simulation method is $\Theta(M(N+E)T) = \Theta(MNT)$. Since the number of outputs of a circuit is typically much smaller than its number of nodes, our proposed method is much more efficient than the full simulation method.

5 EXPERIMENTAL RESULTS

In this section, we present results of various experiments to demonstrate the effect of our proposed batch error estimation method.

5.1 Experiment Setup

To demonstrate the effect of our proposed batch error estimation method, we applied it to an existing ALS method, SASIMI [10]. It identifies pairs of nodes with similar logic function in the circuit and substitutes one node by the other in a pair to reduce the circuit area. It is a greedy iterative ALS flow. In each iteration, it evaluates the error and area reduction for all possible candidate pairs. It then chooses the pair with the highest score calculated as the ratio of area reduction over the error and performs the substitution.

We reimplemented SASIMI approach in our experiment using C++. We used the logic synthesis tool SIS [23] for technology mapping. Since SIS does not consider logic effort of gate during the mapping, the gates are not down-sized when timing requirement is relaxed. Thus, we did not consider the impact of logic downsizing as the original SASIMI. However, we guarantee that the substitution does not increase the circuit delay. We call this SASIMI flow with our proposed batch error estimation technique as *the modified SASIMI*.

All experiments were carried out on a PC with a quad core 3.2GHz CPU (I5-6500) and 32GB RAM. The number of random input patterns used in the MC simulation was 10000 for the experiment in Section 5.2 and 100000 for the others. We assumed that the primary inputs are uniformly distributed. The information of the benchmark circuits used in our experiment can be found in the first three columns of Table 3.

Table 1: Simulated error rate (SER) by MC simulation versus accurate error rate (AER) and simulated average error magnitude (SAEM) by MC simulation versus accurate average error magnitude (AAEM).

alu4		WTM8		MUL8		WTM8	
SER(%)	AER(%)	SER(%)	AER(%)	SAEM	AAEM	SAEM	AAEM
0.390	0.361	0.210	0.244	1.751	1.750	1.863	1.875
0.550	0.549	0.250	0.244	3.784	3.750	3.795	3.797
0.870	0.885	0.570	0.629	7.390	7.437	7.780	8.008
1.120	1.068	1.090	1.010	13.729	13.758	15.589	15.621
3.070	3.033	2.890	2.923	25.574	29.939	29.028	28.839
5.190	5.060	4.310	4.388	59.0417	67.322	63.095	61.595

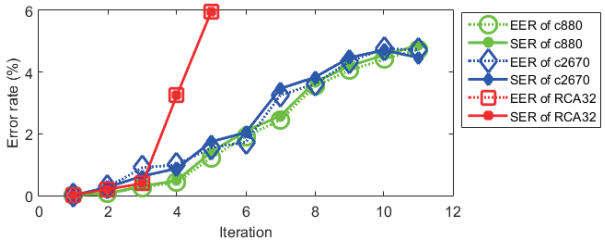

Figure 3: The estimated ER (EER) by our method versus the simulated ER (SER) by MC simulation on three benchmarks.

Table 2: Comparison between the full simulation method and our proposed batch error estimation method.

Circuit	Original	Full simulation		Batch estimation		Speed-up
	area	area	time(s)	area	time(s)	ratio
c880	599	521	14356	521	193	74.4
c1908	1013	663	185863	663	879	211
RCA32	691	664	3561	664	110	32.4

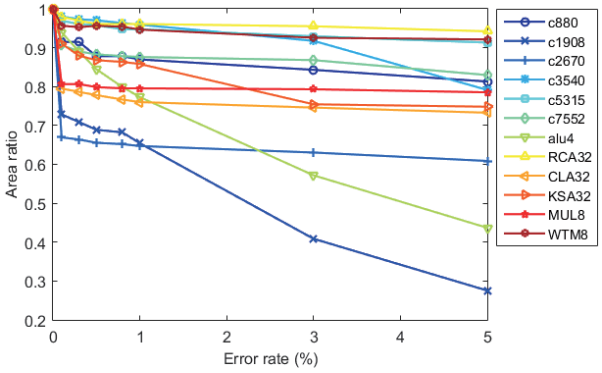

Figure 4: Area ratios of the approximate circuits over the original circuits for different ER thresholds using the modified SASIMI.

5.2 Accuracy of Monte Carlo Simulation

We validate the accuracy of using Monte Carlo simulation in this experiment. Table 1 compares the simulated error obtained by MC simulation with the accurate error obtained by enumeration for some approximate circuits. We consider both ER and AEM. From the table, we can see that although the simulated error by MC simulation and the accurate error are not equal, the difference between them is usually small. Such a small deviation cannot influence the functionality of the approximate circuits seriously due to the error resilience of the target applications.

5.3 Error Estimation Accuracy of Our Method

To demonstrate the accuracy of our batch error estimation method, we compared the estimated ER (EER) by our method and the simulated ER (SER) by the MC simulation for the synthesized approximate circuits for three benchmarks c880, c2670, and a 32-bit adder, RCA32. The results are shown in Fig. 3. EER is the accumulation of the estimated increased ER of the chosen substitution in each iteration. The horizontal axis of the figure gives the iteration number. As shown in Fig. 3, the EER by our method is close to the SER. We can also see that both the EER and the SER of c2670 decrease in the 11-th iteration. This demonstrates that our error estimation method can effectively identify ATs that can reduce the amount of total error. However, EER and SER are not always equal, due to the reconvergent path problem we mentioned in Section 4.3.

5.4 Comparison between the Full Simulation Method and the Proposed Method

To demonstrate the runtime efficiency of our batch error estimation method, we compared it with an accurate full simulation method, in which we simulated the entire fanout cone of the substituted node. This can guarantee the accuracy of error estimation, but it costs a long time. We applied both methods to the original SASIMI [10]. Table 2 shows the runtime and optimization quality comparison between the two methods. Since the full simulation method has an extremely high runtime, we only compared these two methods on three small circuits, which are c880, c1908 and RCA32. We set the error constraint as an ER constraint with threshold of 1%. From the table, we can see that the SASIMI method with our batch error estimation has the same output quality as that with the full simulation method. The last column shows the speed-up ratio of the former over the latter. It can be seen that our method is much faster than the full simulation method. On average, our method achieves a speed-up of 106×.

5.5 Circuit Quality under Error Rate Constraint

We approximated several ISCAS85 benchmarks and arithmetic benchmarks under ER constraint using the modified SASIMI and measured the area ratios of the approximate circuits over the original circuits. Fig. 4 shows how the area ratio changes with the ER threshold. We can see the modified SASIMI can reduce 15%–35% area for most benchmarks under 5% ER threshold. We also compared it with the orignal SASIMI [10] and Wu's method [11] under ER constraint. The results are shown in the last three columns of Table 3. These columns list the average area ratio over seven ER thresholds (0.1%, 0.3%, 0.5%, 0.8%, 1%, 3%, 5%) for each benchmark. Experiment results of SASIMI and Wu's method were taken from the paper [11]. For all the benchmarks, as indicated in bold in the last column, the modified SASIMI gives better quality than both SASIMI and Wu's method. Especially, for the benchmarks alu4, the modified SASIMI can reduce 12% more area than the other two methods. Considering all the benchmarks, the modified SASIMI on average can reduce 4.4% and 3.2% more area than the SASIMI and Wu's method, respectively. In [11], Wu's method was demonstrated to be better than SASIMI. However, with our error estimation method, SASIMI now is better than Wu's. This demonstrates that our error estimation method can truly improve the quality of multi-level ALS flows. The fourth column of Table 3 is the average ratio of the runtime for constructing the CPM over the entire runtime in each iteration. Constructing CPM only occupies 3.5% of total runtime on average. Thus, its runtime overhead is small.

5.6 Circuit Quality under Average Error Magnitude Constraint

We approximated 5 arithmetic benchmarks under the AEM constraint with the modified SASIMI and measured the area ratios of the approximate circuits over the original circuits. Fig. 5 shows the results. The horizontal axis of the figure is the AEM rate, calculated as AEM divided by the maximum binary number encoded by the

Table 3: Comparison among SASIMI [10], Wu's method [11], and the modified SASIMI under ER constraint.

Circuit	Original area	I/O	Ratio of CPM runtime(%)	Average area ratio		
				SASIMI	Wu's	modified SASIMI
c880	599	60/26	4.9	0.896	0.893	**0.873**
c1908	1013	33/25	4.1	0.610	0.595	**0.592**
c2670	1434	233/140	4.8	0.724	0.662	**0.647**
c3540	1615	50/22	2.3	0.975	0.966	**0.936**
c5315	2432	178/123	2.9	0.981	0.978	**0.946**
c7552	2759	207/108	1.3	0.948	0.940	**0.876**
alu4	2740	14/8	2.0	0.892	0.878	**0.751**
RCA32	691	64/33	5.4	0.972	0.970	**0.961**
CLA32	1063	64/33	4.7	0.829	0.822	**0.766**
KSA32	1128	64/33	4.9	0.848	0.849	**0.840**
MUL8	1276	16/16	2.9	0.829	0.819	**0.797**
WTM8	1104	16/16	2.2	0.959	0.953	**0.945**
Arithmean			3.5	0.872	0.860	**0.828**

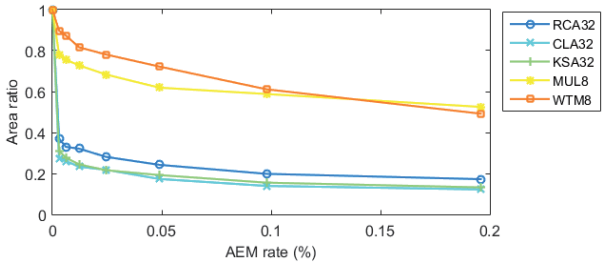

Figure 5: Area ratios of the approximate circuits over the original circuits for different AEM thresholds using the modified SASIMI.

Table 4: Comparison between the original SASIMI [10] and the modified SASIMI under AEM constraint.

Circuit	Original area	Average area ratio	
		SASIMI	modified SASIMI
RCA32	691	0.555	**0.186**
CLA32	1063	0.423	**0.140**
KSA32	1128	0.673	**0.133**
MUL8	1276	0.626	**0.480**
WTM8	1104	0.863	**0.429**
Arithmean		0.628	**0.274**

outputs of a circuit. For AEM less than 0.2% of the maximum value, we can obtain 45%–85% area reduction. We also compared the modified SASIMI with the original SASIMI for different AEM thresholds.[1] Since we used experimental results from paper [10] for the original SASIMI directly, we chose the same AEM thresholds as that paper. Table 4 lists the average area ratio over all AEM thresholds for each benchmark. For all benchmarks, the modified SASIMI has much better results than the original SASIMI, although it does not even apply gate downsizing. On average, the modified SASIMI has an improvement of 2.3× in area over the original SASIMI. The reason why the original SASIMI is much worse than the modified SASIMI is because it only uses the signal probability difference between a pair of internal signals to guide the selection of an AT and it cannot predict the errors at different primary outputs. Therefore, it may choose substitutions that cause errors at the most significant bits, hence reaching the AEM threshold too quickly. This demonstrates that our error estimate method is very helpful in approximating circuits under the AEM constraint.

[1]We did not compare the modified SASIMI with Wu's method, because Wu's method did not consider AEM constraint.

6 CONCLUSION

In this work, we proposed an accurate and efficient batch error estimation method for greedy iterative ALS flows under any statistical error constraint. The key idea is to combine Monte Carlo simulation and local change propagation to estimate the influence of a batch of approximate transformations on all the primary outputs. With our method, ALS approach is more likely to identify local optimal approximate transformations and hence, produce approximate circuits with better quality. We demonstrated theoretically that our method is much more efficient than the full simulation method to obtain errors. We applied our method to an existing ALS method, SASIMI [10]. Experimental results demonstrated that our technique makes the original SASIMI more effective in synthesizing approximate circuits. In our future work, we will extend our method to solve the accuracy loss problem caused by the reconvergent paths. We will also apply the method to other ALS flows, such as the one proposed in [17]. Finally, we will apply the enhanced ALS flows to some real applications.

ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China (NSFC) under Grant No. 61574089.

REFERENCES

[1] M. M. Waldrop. The chips are down for moore's law. *Nature*, 530(7589):144–147, 2016.
[2] J. Han and M. Orshansky. Approximate computing: an emerging paradigm for energy-efficient design. In *ETS*, pages 1–6, 2013.
[3] H. Chung and A. Ortega. Analysis and testing for error tolerant motion estimation. In *DFT*, pages 514–522, 2005.
[4] H. R. Mahdiani, A. Ahmadi, et al. Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(4):850–862, 2010.
[5] K. Y. Kyaw, W. L. Goh, and K. S. Yeo. Low-power high-speed multiplier for error-tolerant application. In *EDSSC*, pages 1–4, 2010.
[6] K. Nepal, Y. Li, et al. ABACUS: A technique for automated behavioral synthesis of approximate computing circuits. In *DATE*, pages 361:1–361:6, 2014.
[7] C. Li, W. Luo, et al. Joint precision optimization and high level synthesis for approximate computing. In *DAC*, pages 104:1–104:6, 2015.
[8] D. Shin and S. K. Gupta. A new circuit simplification method for error tolerant applications. In *DATE*, pages 1–6, 2011.
[9] S. Venkataramani, A. Sabne, et al. SALSA: Systematic logic synthesis of approximate circuits. In *DAC*, pages 796–801, 2012.
[10] S. Venkataramani, K. Roy, and A. Raghunathan. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In *DATE*, pages 1367–1372, 2013.
[11] Y. Wu and W. Qian. An efficient method for multi-level approximate logic synthesis under error rate constraint. In *DAC*, pages 128:1–128:6, 2016.
[12] A. Chandrasekharan, T. Villa, et al. Approximation-aware rewriting of AIGs for error tolerant applications. In *ICCAD*, pages 83:1–83:8, 2016.
[13] Y. Wu, C. Shen, et al. Approximate logic synthesis for FPGA by wire removal and local function change. In *ASPDAC*, pages 163–169, 2017.
[14] Y. Yao, S. Huang, et al. Approximate disjoint bi-decomposition and its application to approximate logic synthesis. In *ICCD*, pages 517–524, 2017.
[15] A. Ranjan, A. Raha, et al. ASLAN: Synthesis of approximate sequential circuits. In *DATE*, pages 364:1–364:6, 2014.
[16] J. Miao, A. Gerstlauer, and M. Orshansky. Multi-level approximate logic synthesis under general error constraints. In *ICCAD*, pages 504–510, 2014.
[17] G. Liu and Z. Zhang. Statistically certified approximate logic synthesis. In *ICCAD*, pages 344–351, 2017.
[18] A. Mishchenko, S. Chatterjee, and R. Brayton. DAG-aware aig rewriting: a fresh look at combinational logic synthesis. In *DAC*, pages 532–535, 2006.
[19] L. Amaru, P. E. Gillardon, and G. D. Micheli. Majority-inverter graph: a new paradigm for logic optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(5):806–819, 2016.
[20] B. Krishnamurthy and I. G. Tollis. Improved techniques for estimating signal probabilities. *IEEE Transactions on Computers*, 38(7):1041–1045, 1989.
[21] R. Ubar, J. Raik, et al. Multiple fault diagnosis with BDD based boolean differential equations. In *BBEC*, pages 77–80, 2012.
[22] T. Y. Hsieh, K. J. Lee, and M. A. Breuer. An error-oriented test methodology to improve yield with error-tolerance. In *VTS*, pages 130–135, 2006.
[23] E. M. Sentovich, K. J. Singh, et al. SIS: A system for sequential circuit synthesis. Technical report, University of California, Berkeley, 1992.