VECBEE: A Versatile Efficiency-Accuracy Configurable Batch Error Estimation Method for Greedy Approximate Logic Synthesis

Sanbao Su, Chang Meng, Fan Yang, Xiaolong Shen, Leibin Ni, Wei Wu, Zhihang Wu, Junfeng Zhao, and Weikang Qian, *Senior Member, IEEE*

***Abstract*—Approximate computing is a recently proposed strategy to improve the energy efficiency of many error tolerant applications. To design approximate circuit automatically, many approximate logic synthesis (ALS) methods have been proposed, among which many are greedy. To improve the synthesis quality of these greedy methods, one key is to calculate the errors of all candidate approximate transformations accurately. However, the traditional simulation-based method is time-consuming. Instead, many existing methods just perform quick but inaccurate error estimation. In this work, to improve both the accuracy and runtime of error estimation, we propose VECBEE, a versatile efficiency-accuracy configurable batch error estimation method for greedy approximate logic synthesis. It is based on Monte Carlo simulation and local change propagation. VECBEE is generally applicable to any statistical error measurement, such as error rate and average error magnitude, and any graph-based circuit representation. The method allows a flexible trade-off between the error estimation accuracy and the runtime, while even the fully accurate version is much faster than the traditional simulation-based method. We applied VECBEE to two representative greedy ALS methods and demonstrated its effectiveness in generating better approximate circuits. The code of VECBEE is made open-source.**

***Index Terms*—approximate computing, approximate logic synthesis, error estimation**

I. Introduction

As the transistor size shrinks into the nano-scale [1], power consumption has become a major concern in designing modern computing systems. Meanwhile, much workload of computing systems today are error tolerant applications, such as machine learning, data mining, and image processing, to name a few. For example, in image processing, a few erroneous pixels would not affect human recognition of the images [2]. Given these trends, *approximate computing* [2]–[4] was proposed as a novel power-efficient design paradigm for these error tolerant applications. Its basic idea is to deliberately introduce a small amount of error into the computing systems. If the error is introduced properly, significant improvement in area, delay, and power consumption can be achieved.

The concept of approximate computing is applicable to almost all layers of modern computing systems. At the circuit layer, there are two main research fields: manual design and automatic synthesis. The former manually designs some widely-used approximate arithmetic units such as adders [5]–[9] and multipliers [10]–[14]. The latter develops algorithms to produce a good approximate version for an arbitrarily given circuit. It can be further divided into approximate high-level synthesis [15]–[17] and *approximate logic synthesis (ALS)* [18]–[32]. In ALS, many studies target at the common circuit form, the multi-level circuit [20]–[32].

To explore the extremely large design space of an approximate circuit efficiently, many multi-level ALS methods [20]–[26], [30] are greedy. They derive the final approximate circuit through multiple rounds of *approximate local transformations (ALTs)*. In each round, all candidate ALTs are identified. Then, the quality improvement such as area, delay, or power improvement and the induced error such as *error rate (ER)* or *average error magnitude (AEM)* of each ALT are evaluated. The one with the largest *figure-of-merit (FOM)*, which is typically calculated as the ratio of the quality improvement over the error, is then selected and applied.

For these methods, it is critical to calculate the induced error accurately. Otherwise, the greedy method may choose an inferior ALT in each round, which eventually leads to an inferior final approximate design. Even worse, an inaccurate error estimation may overestimate the errors, causing a premature termination of the ALS loop. This is because at some round, the overestimated errors of the candidate ALTs all exceed the error bound and the loop stops at this round, but with accurate error estimation, the loop can still continue.

A motivating example is given in Fig. 1. It shows the importance of the accurate error estimation to an existing ALS method *SASIMI* [22]. The red and blue curves show how the SASIMI methods with and without accurate error estimation, respectively, work on the benchmark c7552. The error constraint is an ER threshold of 1%. Each point on a curve corresponds to the result after one round in the ALS flow. The blue curve shows that the method without accurate

Sanbao Su (gawaine@sjtu.edu.cn), Chang Meng (changmeng@sjtu.edu.cn), and Weikang Qian (qianwk@sjtu.edu.cn) are with the University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, China. Weikang Qian is also with the MoE Key Lab of Artificial Intelligence, Shanghai Jiao Tong University, China and the State Key Laboratory of ASIC & System, Fudan University, China.

Fan Yang (yangfan@fudan.edu.cn) is with the School of Microelectronics and the State Key Laboratory of ASIC & System, Fudan University, China.

Xiaolong Shen (shenxiaolong3@huawei.com), Leibin Ni (nileibin@huawei.com), Wei Wu (wuwei102@huawei.com), Zhihang Wu (wuzhihang@huawei.com), and Junfeng Zhao (junfeng.zhao@huawei.com) are with Huawei Technologies Co., Ltd.

This work is supported in part by the State Key Laboratory of ASIC & System Open Research Grant 2019KF004 and the National Natural Science Foundation of China (NSFC) Grant 61574089.

Sanbao Su and Chang Meng contributed equally. Corresponding author: Weikang Qian.

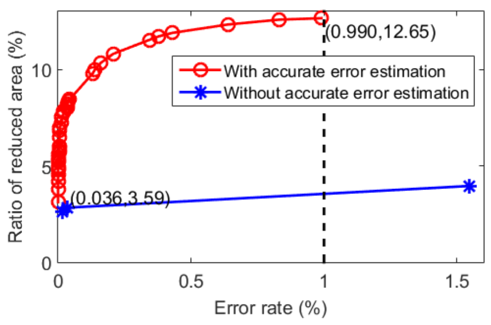

Fig. 1. Ratio of area reduction versus error rate for the same ALS method SASIMI with and without accurate error estimation. The benchmark is c7552 and the error rate threshold is 1%.

error estimation stops after 3 iterations. This is because at the third iteration, due to the inaccurate error estimation, the best candidate ALT it selects actually increases the ER by 1.5%, which causes the resulting circuit to exceed the error limit. In contrast, with accurate error estimation, candidate ALTs 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 9.06% more area than that without accurate error estimation.

The above example clearly shows that an accurate error estimation is helpful to a greedy ALS method. The traditional method for accurate error estimation needs to apply each ALT to the current circuit and simulate the resulting circuit. We refer to this method as *traditional simulation-based method* in this paper. However, this method is time-consuming, since in order to get the errors for all the ALTs, its number of simulations equals the number of ALTs. Consequently, many existing greedy ALS methods just perform quick but inaccurate error estimation.

To improve the quality of a greedy ALS approach without increasing its runtime, we need to develop an efficient and accurate batch error estimation method for all the candidate ALTs. For this purpose, in this work, we propose *VECBEE*, a versatile efficiency-accuracy configurable batch error estimation method for greedy ALS flows. To make it generally applicable to any input distribution, VECBEE is based on *Monte Carlo (MC)* simulation. To avoid the time-consuming MC simulation for all the ALTs, we develop an efficient local change propagation technique that can capture the effect of an internal approximation-induced error on all the *primary outputs (POs)*. VECBEE allows a trade-off between efficiency and accuracy. Although the most efficient version is not fully accurate, it is still accurate enough. The fully accurate version takes a longer runtime, but it is still much faster than the traditional simulation-based method.

VECBEE is versatile. It is applicable to any statistical error measure, such as ER and AEM, and any input distribution. It can also be applied to any graph-based representation of circuits, such as *AND-inverter graph (AIG)* [33], *majority-inverter graph (MIG)* [34], and gate netlist after technology mapping. Yet, one requirement for applying VECBEE to a greedy ALS flow is that the local circuits affected by the ALTs have a single output. Fortunately, many existing ALTs satisfy this requirement.

We apply VECBEE to two representative ALS methods, SASIMI [22] and approximate node simplification (ANS) [23]. Our experiment results showed that the ALS methods enhanced by VECBEE improved the circuit quality for both the ER and the AEM constraints compared to the original methods. The code of VECBEE is made open-source at https://github.com/SJTU-ECTL/VECBEE.

A preliminary version of this work is published in [35]. In that work, we propose a basic batch error estimation method, which is efficient but not fully accurate. Its effectiveness is demonstrated by combining it with the SASIMI ALS flow. In this work, we further extend the basic method to make it efficiency-accuracy configurable. Particularly, the improved method includes the fully accurate mode. We also apply the proposed method to the ANS ALS flow to demonstrate its wide applicability.

The rest of this paper is organized as follows. Section II discusses the related works. Section III presents the preliminaries. Section IV describes the VECBEE methodology in detail. Section V shows the experimental results. Finally, Section VI concludes the paper.

II. Related Works

In this section, we first briefly discuss some representative ALS methods for multi-level circuits and point out whether VECBEE is applicable.

The work [21] proposed a systematic ALS method that encodes the error constraint through a circuit and utilizes external don't cares for simplification. Its error constraint is maximum error magnitude, which is not a statistical error measure. Thus, VECBEE is not applicable. The work [36] proposed an ALS method with ALTs based on *Boolean matrix factorization (BMF)*. Although it is a greedy ALS flow, its ALTs are applied to sub-circuits that can have multiple outputs. Since VECBEE can only deal with ALTs applied to local circuits with a single output, it is not applicable to this method. The work [29] proposed an ALS approach with stochastic optimization. In each iteration, it randomly selects an ALT and accepts it probabilistically. For early iterations, VECBEE may not help because there is only one ALT under consideration and hence, an accurate error calculation for this single ALT is affordable. However, we may apply VECBEE in later iterations when the accumulated error is near the limit. In this case, due to the reduced error margin, it may be advantageous to consider multiple candidate ALTs and choose the best one.

Several other works proposed greedy ALS methods for statistical error measure [20], [22], [23], [25]. Thus, VECBEE can help improve their synthesis quality or algorithm runtime.

In [20], Shin and Gupta proposed a greedy ALS method under ER and error magnitude constraint. Its ALT is approximating a signal with a constant 0 or 1. ER is estimated by parallel fault simulation with random input vectors. VECBEE can accelerate the error estimation and hence, speed up the entire ALS method.

In [22], Venkataramani *et al.* presented a greedy ALS approach, SASIMI, which can handle either ER or AEM constraint. Its ALT is substituting a wire in the circuit with another of similar functionality. To avoid the time-consuming error estimation at the POs, the error for each substitution is estimated as the probabilities that the two signals in the substitution pair are different. VECBEE can improve the error estimation accuracy and hence, improve the synthesis quality of SASIMI.

In [23], Wu and Qian proposed a greedy ALS methods under the ER constraint. It works on Boolean network representation of the circuit. Its ALT is an approximate node simplification (ANS), which deletes some literals from the factored-form expression of a node in the Boolean network. The ER of each ALT is estimated as the ER at the output of each changed node, not the ER at the POs. To improve the estimation accuracy, the effect of don't cares is considered. However, their approach can only compute an upper bound of the ER, which may be much larger than the accurate ER. VECBEE can be applied to estimate the ER more accurately and improve the synthesis quality of their ALS method.

The work [25] proposed a greedy ALS approach for FPGA designs under ER constraint. Its ALT is removing one input of a single-output local circuit and then reconfiguring the local function. Similar to [23], it estimates ER based on the output of the local circuit, not the POs. Thus, its ER estimation is inaccurate. Again, VECBEE can help improve its synthesis quality.

Finally, we point out that there are two recent works also focusing on the error analysis of ALTs. In [37], Scarabottolo *et al.* proposed to partition the circuit into sub-circuits, and then determine the error propagation model of the resulting sub-circuits. However, it analyzes the maximum error magnitude, which is different from the statistical error metrics considered in our work. In [38], Echavarria *et al.* proposed an error transition model that propagates the bit error rate through cascaded sub-circuits. Such a method does not rely on logic simulation, instead, it directly propagates the signal distributions. Nevertheless, its accuracy is affected by reconvergent paths and the error magnitude metric is not supported.

III. Preliminaries

We introduce some preliminaries in this section.

A. Circuit Terminology

We focus on multi-level combinational circuits, which can be modeled as a directed acyclic graph with nodes representing input pins and logic gates, and directed edges representing wires connecting the gates. *Source nodes* are those nodes in the graph without any fanins, while *sink nodes* are those nodes in the graph without any fanouts. The *primary inputs (PIs)* are source nodes of the graph. The *POs* are a subset of the nodes of the graph, including all the sink nodes and possibly some non-sink nodes of the graph.

In a circuit, if there is a path from node n to node m, then m is a *transitive fanout (TFO)* of n and n is a *transitive fanin (TFI)* of n. Node n itself is a trivial TFO/TFI of n. The *TFO (resp. TFI) cone* of n is a node set that includes all the TFOs (resp. TFIs) of n.

B. Statistical Error Measure

In this work, we consider the situation where statistical error measure is used. Two typical statistical error measures are **error rate (ER)** and **average error magnitude (AEM)**. We use $\vec{x}$ to denote an input vector of a circuit and use $\overrightarrow{f_{org}}(\vec{x})$ and $\overrightarrow{f_{app}}(\vec{x})$ to denote output vectors of the original and approximate circuits for the input vector $\vec{x}$, respectively. ER is defined as

$$ER = \sum_{\vec{x}:\overrightarrow{f_{app}}(\vec{x}) \neq \overrightarrow{f_{org}}(\vec{x})} P(\vec{x}), \quad (1)$$

where $P(\vec{x})$ is the occurrence probability of the input vector $\vec{x}$. AEM is defined as

$$AEM = \sum_{\vec{x}:\overrightarrow{f_{org}}(\vec{x}) \neq \overrightarrow{f_{app}}(\vec{x})} \left| \overrightarrow{f_{app}}(\vec{x}) - \overrightarrow{f_{org}}(\vec{x}) \right| P(\vec{x}), \quad (2)$$

where $|\overrightarrow{f_{app}}(\vec{x}) - \overrightarrow{f_{org}}(\vec{x})|$ represents the absolute difference between the binary number encoded by $\overrightarrow{f_{app}}(\vec{x})$ and that by $\overrightarrow{f_{org}}(\vec{x})$. AEM is usually applied to arithmetic circuits.

C. Greedy ALS Methods

As we stated in Section II, many existing multi-level ALS methods are greedy. They can be summarized into a general procedure shown in Algorithm 1. Its basic idea is to gradually improve the circuit by applying an ALT in each iteration. An ALT is a small perturbation to the circuit. Several examples of ALT can be found in Section II.

In each iteration, the procedure chooses the optimal ALT and applies it to the current approximate circuit C_{app}. For this purpose, all possible ALTs of C_{app} will first be collected (see Line 4). Then, the quality improvement ΔQ and the error increase ΔE caused by each ALT are evaluated (see Lines 6–7), where the quality could be area, delay, or power consumption and the error could be ER or AEM. Note that both the quality improvement and the error increase for an ALT are calculated over the current approximate circuit. Then, the ALT that maximizes an FOM defined over ΔQ and ΔE is selected and applied (see Line 8). A typical FOM is $\Delta Q/\Delta E$, which favors an ALT that maximizes the quality improvement while minimizing the error increase. The final step of each iteration is to calculate the actual error of the new approximate circuit (see Line 9). If the actual error is smaller than the given error threshold E_{th}, then the next iteration begins; otherwise, the entire loop finishes and the latest approximate circuit is returned.

IV. Methodology

As we stated in Section I, an efficient and accurate batch error estimation method is critical to a greedy ALS method. In this section, we describe VECBEE, the proposed method for an efficient and accurate batch error estimation.

Algorithm 1: A general greedy ALS procedure.

```
Input  : original circuit $C$ and error threshold $E_{th}$
Output : approximate circuit $C_{app}$
1  error $E \leftarrow 0$; new circuit $C_{new} \leftarrow C$;
2  while $E \leq E_{th}$ do
3      $C_{app} \leftarrow C_{new}$;
4      identify candidate ALT set $S$ from $C_{app}$;
5      foreach ALT $A$ in $S$ do
6          estimate quality improvement $\Delta Q$ due to $A$;
7          estimate error increase $\Delta E$ due to $A$;
8      choose the ALT with the highest FOM $f(\Delta Q, \Delta E)$ and
         apply it to generate new approximate circuit $C_{new}$;
9      calculate accurate error $E$ between $C_{new}$ and $C$;
10 return $C_{app}$
```

A. Overview of VECBEE

VECBEE 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 or AEM, which is of interest in this work, there are usually two ways to obtain it, analytical methods and MC simulation. The analytical methods are based on signal probability propagation [39] or binary decision diagram (BDD) [40]. They only work when all the inputs are independent, which may not be true for a general input distribution. Furthermore, its scalability is a problem. Thus, to make the approach more general, MC-based logic simulation should be applied. Strictly speaking, an MC simulation cannot give the exact result due to its random variation. However, by the law of large numbers, if a sufficient number of samples are used, the final obtained result will be very close to the exact value [41].

From the general procedure shown in Algorithm 1, an important step is to do batch evaluation of the errors for all candidate ALTs in one iteration. To obtain the accurate error for *each* ALT, the circuit $C_{App,ALT}$ obtained by applying that ALT 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 the candidate ALTs, the total number of MC simulations equals the number of ALTs, which causes a long runtime.

To reduce the runtime, some previous methods just use the error observed at the output of the local circuit affected by the ALT as an estimate to the final exact error [22], [23], [25]. In this case, for each ALT, we only need to simulate the local circuit affected by the ALT by propagating existing simulation results at its inputs to its outputs. We do not need to propagate the simulation results to the POs of the circuit. Thus, simulation time can be significantly reduced. However, since these methods ignore the potential logic masking effect from the output of a local circuit to the POs, the error estimation could be quite inaccurate. Below shows an example.

Example 1
Consider the circuit shown in Fig. 2. Nodes $I_1, I_2, \ldots, I_7$ are the PIs and nodes O_1, O_2 are the POs. Assume that the i-th input pattern in the MC simulation is $I_1 I_2 \ldots I_7 = 0111011$. The value for each wire under this input pattern is shown above the wire in the figure. Consider an ALT that simply replaces node e with a constant 0. Under the current input pattern, this ALT causes an error at node e. However, the error is masked by the following NOR gate given that $I_7 = 1$. Thus, the error cannot be propagated to PO O_2. Besides, PO O_1 does not depend on node e. Thus, for that input pattern, the error at node e does not cause an output error for the circuit. □

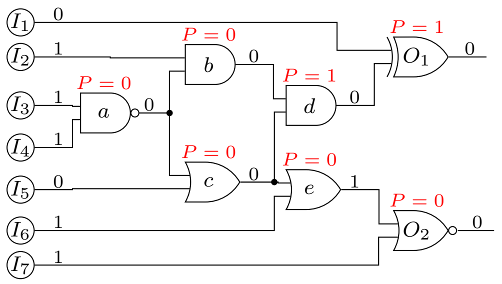

Fig. 2. An example circuit. The number above each wire is the signal value under the i-th input pattern in the MC simulation. The red number above each gate n is the change propagation matrix entry $P[i, n, O_1]$ computed by the method in Section IV-B.

In this work, we propose a method to enhance the accuracy of batch error estimation while still using a small amount of MC simulation. The procedure of our proposed method, VECBEE, is shown in Algorithm 2. It gives the error increases for all the ALTs of the current approximate circuit.

Algorithm 2: The proposed batch error estimation procedure, VECBEE, for one iteration in an ALS flow.

```
Input  : the original circuit $C_{org}$, the current approximate
         circuit $C_{app}$, and the sample number $M$ in the
         MC simulation
Output : the vector $\Delta E$ of error increases for all the ALTs
         of $C_{app}$
1  generate the matrix $\Omega$ representing $M$ random input patterns;
2  simulate $C_{org}$ with $\Omega$ and get the output value matrix $O_{org}$;
3  simulate $C_{app}$ with $\Omega$ and get the node value matrix $N_{app}$;
4  $P \leftarrow$ GetCPM($C_{app}$, $N_{app}$);
5  identify candidate ALT set $S$ from $C_{app}$;
6  foreach ALT $A$ in $S$ do
7      $\Delta E[A] \leftarrow$ ErrorEstimate($A$, $C_{app}$, $P$, $O_{org}$, $N_{app}$);
8  return $\Delta E$
```

Assume the number of samples used in the MC simulation is M, the number of outputs of the original circuit is O, and the number of nodes of the current approximate circuit is N. VECBEE first generates M random input patterns, which are sampled from a given input distribution (see Line 1). It then applies them to both the original circuit C_{org} and the current approximate circuit C_{app} (see Lines 2–3). It produces the output value matrix O_{org} of the original circuit and the node value matrix N_{app} of the current approximate circuit, which will be used later. The matrix O_{org} is an $M \times O$ matrix. It records the values for all the POs of C_{org} in the simulation. Each entry in O_{org} is indexed as $O_{org}[i, o]$, where $1 \leq i \leq M$ corresponds to the i-th input pattern and o represents a PO of the circuit C_{org}. The entry $O_{org}[i, o]$ gives the value of PO o under the i-th input pattern. The matrix N_{app} is an $M \times N$ matrix. It records the values for all the nodes of

C_{app} in the simulation. Each entry in N_{app} is indexed as $N_{app}[i,n]$, where $1 \leq i \leq M$ corresponds to the i-th input pattern and n represents a node in the approximate circuit C_{app}. The entry $N_{app}[i,n]$ gives the value of node n under the i-th input pattern.

The key steps of VECBEE are shown by Lines 4–7 in Algorithm 2. In order to obtain the error increase caused by each ALT, we propose to first characterize whether a value change occurring at the output of the local circuit affected by the ALT will be propagated to each PO. To capture the local change propagation, we define a change propagation matrix.

Definition 1
A **change propagation matrix (CPM)** *P for a circuit is a three-dimensional 0-1 matrix of size $M \times N \times O$. Each entry in the CPM is indexed as $P[i,n,o]$, where $1 \leq i \leq M$ corresponds to the i-th sample in the MC simulation, n represents a node in the circuit, and o represents a PO of the circuit. If the entry $P[i,n,o] = 1$, it means a value change on node n can be propagated to the output o under the i-th input pattern; otherwise, it cannot.* □

Line 4 obtains the CPM for the current approximate circuit C_{app} based on the node value matrix N_{app} obtained from the MC simulation. Two detailed implementations of this step will be shown in Sections IV-B and IV-C, respectively. Specifically, Section IV-B shows a very efficient method to obtain CPM. However, it is not guaranteed to be always accurate. Section IV-C extends that method and derives an efficiency-accuracy configurable version, which can be configured into a fully accurate version at the cost of longer runtime. After obtaining the CPM, the procedure estimates the error increase for each ALT of C_{app} based on the CPM, the output value matrix O_{org}, and the node value matrix N_{app} (see Line 7). The details of this step will be described in Section IV-D. Note that with the help of the CPM, estimating the error increases for all the ALTs requires no additional MC simulation of the entire circuit. Finally, a time complexity analysis of VECBEE is given in Section IV-E.

B. An Efficient Approximate Method to Obtain the Change Propagation Matrix

In this section, we present a very efficient method to obtain the CPM. However, it is subject to some accuracy loss. The method is based on Boolean difference, defined as follows.

Definition 2
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:*

$$\frac{\partial f}{\partial x} = f_x \oplus f_{\bar{x}}, \tag{3}$$

where f_x and $f_{\bar{x}}$ are the **positive cofactor** *and the* **negative cofactor** *of f with respect to x, 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 definition, we can see that if a combination of all the other input variables lets the BD $\frac{\partial f}{\partial x}$ be 1, then for this combination, the value of f will change if x changes.

For each node n in the circuit, we define S_n as the set of direct fanouts of node n. For each $1 \leq i \leq M$, each node n in the circuit, 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 $D[i,n,n_f]$, which can be obtained by two steps. First, we compute the BD $\frac{\partial n_f}{\partial n}$ by Eq. (3). 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 and get the value $D[i,n,n_f]$.

Example 2
In Fig. 2, the function of node O_2 is $\overline{e + I_7}$. By Eq. (3), the BD of O_2 with respect to e is

$$\frac{\partial O_2}{\partial e} = \overline{1 + I_7} \oplus \overline{0 + I_7} = \overline{I_7}.$$

Since $I_7 = 1$ under the i-th input pattern, we have that $D[i,e,O_2] = 0$. Similarly, we can obtain that $D[i,d,O_1] = 1$, $D[i,c,d] = 0$, and $D[i,c,e] = 0$. □

If $D[i,n,n_f] = 1$, it means that a value change on n will be propagated to n_f under the i-th input pattern.

Now, we show how to obtain the CPM. For each fixed i and fixed PO o, we will get $P[i,n,o]$ for each node n in the circuit. We first handle the case where n is just o. Since a value change on o can always be observed at o itself, we have $P[i,o,o] = 1$. Then, we obtain the values of $P[i,n,o]$ for all the remaining nodes n in the circuit in a reverse topological traversal over the circuit. We start from the sink nodes, such as O_1 and O_2 in Fig. 2. For any sink node n except o, since it can never have PO o in its TFO cone, its value change cannot be observed at o. Thus, we have $P[i,n,o] = 0$. For any other node n, the value $P[i,n,o]$ can be recursively calculated. Indeed, a value change on n can be observed at the output o when n has a fanout n_f such that the value change on n causes a value change on n_f and the latter can be propagated to the output o. Note that if a value change on n causes a value change on n_f, we must have $D[i,n,n_f] = 1$. If a value change on n_f can be propagated to the output o, we must have $P[i,n_f,o] = 1$. Thus, we can conclude the following recursive formula for calculating $P[i,n,o]$:

$$P[i,n,o] = \bigvee_{\forall n_f \in S_n} (P[i,n_f,o] \wedge D[i,n,n_f]), \tag{4}$$

where S_n is the set of direct fanouts of node n.

Example 3
Consider the circuit in Fig. 2. Suppose that we want to get the CPM entry $P[i,c,O_1]$ for the i-th simulation sample. Initially, we have $P[i,O_1,O_1] = 1$. For the sink node O_2, we have $P[i,O_2,O_1] = 0$. By applying Eq. (4) in a reverse topological order and using the BD values from Example 2, we can obtain

$$\begin{aligned} P[i,d,O_1] &= P[i,O_1,O_1] \wedge D[i,d,O_1] = 1, \\ P[i,e,O_1] &= P[i,O_2,O_1] \wedge D[i,e,O_2] = 0, \\ P[i,c,O_1] &= (P[i,d,O_1] \wedge D[i,c,d]) \\ &\quad \vee (P[i,e,O_1] \wedge D[i,c,e]) = 0. \end{aligned}$$

Thus, a value change on c cannot be propagated to O_1 for the i-th simulation sample. □

The procedure of computing the CPM is shown in Algorithm 3. It takes a circuit C and a node value matrix N_v obtained from an MC simulation as inputs. Line 2 computes the $D[i, n, n_f]$ values based on the BD function and the local input values of each node n. Line 5 sets the CPM entry $P[i, o, o]$ to 1. Lines 6–7 set the CPM entry $P[i, n, o]$ to 0 for each sink node n in the circuit except node o. Lines 8–13 compute the values $P[i, n, o]$ according to Eq. (4) for all the remaining nodes in the circuit following the reverse topological order.

Algorithm 3: The function *GetCPM*(C, N_v) for computing the change propagation matrix P.

```
Input  : a circuit $C$ and a node value matrix $N_v$ obtained
         from MC simulation
Output : three-dimensional change propagation matrix $P$
1  $M \leftarrow$ number of samples in the MC simulation;
2  compute $D[i, n, n_f]$ from the node value matrix $N_v$ for all
     $1 \le i \le M$, nodes $n$ in $C$, and fanouts $n_f$ of node $n$;
3  foreach PO $o$ in $C$ do
4      for $i$ from 1 to $M$ do
5          $P[i, o, o] \leftarrow 1$;
6          foreach sink node $n$ in $C$ except node $o$ do
7              $P[i, n, o] \leftarrow 0$;
8      foreach node $n$ in $C$ except node $o$ and sink nodes in
         reverse topological order do
9          $S_n \leftarrow$ the set of direct fanouts of node $n$;
10         for $i$ from 1 to $M$ do
11             $P[i, n, o] \leftarrow 0$;
12             foreach node $n_f$ in $S_n$ do
13                 $P[i, n, o] \leftarrow$
                     $P[i, n, o] \vee (P[i, n_f, o] \wedge D[i, n, n_f])$;
14 return $P$
```

However, the obtained CPM sometimes is inaccurate due to the existence of reconvergent paths. The following shows an example.

Example 4
In Fig. 2, node a can reach PO O_1 through either the path $a \to b \to d \to O_1$ or the path $a \to c \to d \to O_1$. By applying Algorithm 3, we have $P[i, a, O_1] = 0$, which means that a value change on a cannot propagate to O_1 under the i-th input pattern. However, this CPM value is inaccurate. If we change a's value from 0 to 1 and simulate the circuit again, we will have $a = 1$, $b = 1$, $c = 1$, $d = 1$, and $O_1 = 1$. Since $O_1 = 0$ originally, a value change on a is propagated to O_1 under the i-the input pattern. Thus, the correct value of $P[i, a, O_1]$ should be 1. □

The reason why the existence of reconvergent paths causes failure of Eq. (4) in Example 4 can be understood as follows. By Eq. (4), $P[i, a, O_1]$ depends on $P[i, b, O_1]$, which further depends on $D[i, b, d]$. This dependency is essentially due to the path $a \to b \to d \to O_1$. Since $\frac{\partial d}{\partial b} = c$, $D[i, b, d]$ is the value of c under the i-th input pattern. Due to the existence of another path $a \to c \to d \to O_1$, the value of c depends on the value of a. However, during the recursive procedure to calculate $P[i, a, O_1]$, we use the value of c before a changes. This causes a wrong value for $D[i, b, d]$ and hence, a wrong value for $P[i, a, O_1]$.

C. Improving the Accuracy of Change Propagation Matrix Calculation: An Efficiency-Accuracy Configurable Approach

From Example 4, we can see that the existence of reconvergent paths can cause the inaccuracy of the CPM calculation. To address this issue, one solution is to minimize the effect of the reconvergent paths. We first show how to obtain the fully accurate CPM. Then, we present a method to trade accuracy for runtime efficiency.

1) Computation of Fully Accurate CPM: In order to introduce our method for computing a fully accurate CPM, we first give the following definitions.

Definition 3
Assume that node n_2 is a TFO of node n_1. The **sub-circuit** *between nodes n_1 and n_2 is a sub-circuit consisting of the nodes and edges on all paths from n_1 to n_2.* □

For example, the sub-circuit between nodes a and O_1 in Fig. 2 consists of nodes a, b, c, d, and O_1 and the edges connecting them, since there are two paths from a to O_1, $a \rightarrow b \rightarrow d \rightarrow O_1$ and $a \rightarrow c \rightarrow d \rightarrow O_1$. By definition, the sub-circuit between nodes n_1 and n_2 can be obtained by intersecting the TFO cone of n_1 and the TFI cone of n_2.

Definition 4
The **one-cut** *between a node n and a PO o is a node t not equal to n and closest to n satisfying that all paths from n to o pass through t.* □

For example, in Fig. 2, the one-cut between node a and PO O_1 is node d, while the one-cut between node a and PO O_2 is node c.

Assume that the one-cut t between a node n and a PO o has been determined. Then, we can derive the CPM entries $P[i,n,o]$ for each i. The one-cut t between a node n and o divides the sub-circuit between n and o into two sub-circuits. The first is the sub-circuit between n and t and the second is the sub-circuit between t and o. By the property of the one-cut t, all paths from n and o pass through t. Therefore, if a value change on n propagates to o (i.e., $P[i,n,o]=1$), then that change should propagate to t through the first sub-circuit (i.e., $D[i,n,t]=1$, where $D[i,n,t]$ is the BD of t with respect to n under the i-th input pattern), and the value change on t should further propagate to o through the second sub-circuit (i.e., $P[i,t,o]=1$). Thus, we can calculate $P[i,n,o]$ as follows:

$$P[i,n,o] = P[i,t,o] \wedge D[i,n,t]. \tag{5}$$

We apply Eq. (5) to calculate entries $P[i,n,o]$ following a reverse topological order on all the nodes in the circuit. Thus, when we apply the equation to compute $P[i,n,o]$ for a particular n, $P[i,t,o]$ has already been obtained. The remaining task is to obtain the BD $D[i,n,t]$. It is done by simulating the sub-circuit between n and t. Specifically, under the i-th PI pattern, we flip the value of n and re-simulate the sub-circuit between n and t. If the value of t changes, we have $D[i,n,t]=1$; otherwise, $D[i,n,t]=0$.

Example 5
In Fig. 2, the one-cut between a and O_1 is d. To compute $D[i,a,d]$ under input pattern i, we change the value of a from 0 to 1 and then simulate the sub-circuit between a and d. In sequence, we have $a=1$, $b=1$, $c=1$, and $d=1$. Thus, the value change of a propagates to d and hence, $D[i,a,d]=1$. Since $P[i,d,O_1]=1$, by Eq. (5), we have $P[i,a,O_1]=P[i,d,O_1] \wedge D[i,a,d]=1$. Compared to Example 4, the new approach obtains an accurate CPM entry. □

The last problem is how to find the one-cut between a node n and a PO o. A network flow-based approach is designed to solve the problem. Initially, the sub-circuit G_{sub} between n and o is extracted. We treat the node n as the source node with an initial incoming flow of 1. The flow starts from the source node n and propagates through nodes in the sub-circuit G_{sub} in a topological order. Each node has a **total incoming flow** and each edge has an associated flow. The total incoming flow for a node except n is the sum of the flows on its incoming edges, while the flow on an edge equals the total incoming flow of its source node divided by the number of fanouts of the source node. In other words, the total incoming flow of a node is distributed evenly among the outgoing edges of the node. The first node except n in the topological order with a total incoming flow of 1 is the one-cut between node n and o. Note that the total incoming flow of 1 guarantees that all paths from n to o must pass through the node.

Example 6
Suppose that we want to obtain the one-cut between node a and PO O_1 in Fig. 2. The sub-circuit between a and O_1, G_{sub}, consists of nodes a, b, c, d, and O_1 in a topological order. The incoming flow to node a is set as 1. That flow is distributed evenly over the two outgoing edges of a. Thus, the flows on edges (a,b) and (a,c) are both $1/2$. Within the sub-circuit G_{sub}, b has only one incoming edge (a,b) and hence, the total incoming flow of b is $1/2$. The same for node c. Within the sub-circuit G_{sub}, node b has a single outgoing edge (b,d). Thus, the flow on that edge is $1/2$. Similarly, the flow on edge (c,d) is $1/2$. Finally, we can get the total incoming flow of node d as 1. Since d is the first node except a in the topological order with a total incoming flow of 1, it is the one-cut between a and O_1. □

2) Trade-off Between Efficiency and Accuracy: In the last subsection, we present an approach to obtain the CPM accurately. However, it can consume a long time. To compute the exact CPM entry $P[i,n,o]$ by Eq. (5), it needs to calculate the BD value $D[i,n,t]$ by flipping the value of node n and simulating the sub-circuit between node n and the one-cut t. When t is far from n[1], it is time-consuming to simulate the sub-circuit. To avoid simulating a complex sub-circuit with a large logic depth, we set a depth limit l on the sub-circuit. If the logic depth between n and t is no more than l, Eq. (5) is applied to compute an exact CPM entry. Otherwise, the following equation is used to compute the entry approximately:

$$P[i,n,o] = \bigvee_{\forall n_f \in S'_{n,l}} (P[i,n_f,o] \wedge D[i,n,n_f]), \tag{6}$$

[1]The worst case happens when t is the PO o.

where $S'_{n,l}$ is the l**-th level boundary** of node n.

Generally speaking, the l-th level boundary $S'_{n,l}$ of node n is a set of nodes in n's TFO cone such that the logic levels of all the nodes are at least l. To produce the l-th level boundary, we first extract the TFO cone of n and relabel the logic levels of all nodes in the cone. The level of a non-PO node k in the cone is the length of the longest path from n to k, while the level of a PO is labelled as infinity. The 1st level boundary of n consists of the direct fanouts of n. To obtain the l-th ($l > 1$) level boundary, a node set S is initialized with the 1st level boundary of n. The levels of all nodes in the l-th level boundary are required to be at least l. If the requirement is not satisfied for a node in S, then a node u with the smallest level in S is replaced by all of its fanouts. This update is repeated until the levels of all nodes in S are at least l. At this moment, the obtained node set S is the l-th level boundary.

Example 7

Fig. 3 shows the TFO cone of node a, where each circle represents a gate and each number is the logic level of the corresponding gate. Nodes o_1 and o_2 are POs and their logic levels are set to infinity. By definition, the 1st level boundary of a is formed by the direct fanouts of a. Thus, $S'_{a,1} = \{b, c, d\}$. To obtain the 2nd level boundary of a, nodes b and c in $S'_{a,1}$, which are with levels less than 2, are replaced by their direct fanouts d, e, and o_2, all of which are with levels at least 2. Thus, the 2nd level boundary of a is $S'_{a,2} = \{d, e, o_2\}$. To obtain the 3rd level boundary of a, node d in $S'_{a,2}$ is replaced with its direct fanout e, which is with level 3. Thus, the 3rd level boundary of a is $S'_{a,3} = \{e, o_2\}$. □

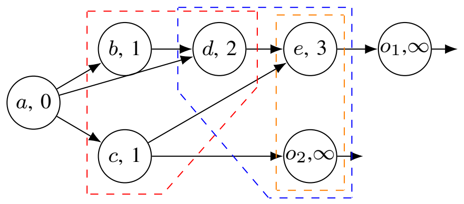

Fig. 3. Examples of l-th level boundary. The red, blue, and orange polygons circle out the 1st, 2nd, and 3rd level boundary of a, respectively.

The depth limit l can tune the accuracy of CPM. When $l = 1$, Eq. (6) degrades into Eq. (4), because $S'_{n,1}$ only contains the direct fanouts of n. As l becomes larger, nodes in $S'_{n,l}$ are further away from n, as shown in Example 7. Therefore, more reconvergent paths starting from node n converge before the l-th level boundary and hence, the accuracy of CPM computed by Eq. (6) increases. When l becomes sufficiently large, then Eq. (5) is applied to compute every entry in the CPM, and the CPM becomes fully accurate.

Based on the above discussion, we can update Algorithm 3 into an efficiency-accuracy configurable version, which is shown in Algorithm 4. It takes as inputs a circuit C, a node value matrix N_v obtained from MC simulation, and a depth limit l. It outputs the CPM P. For each PO o, Lines 4–6 first initialize CPM entries for PO o and the sink nodes. For each node n except PO o and the sink nodes in reverse topological order, Line 8 first obtains the one-cut t between n and o. If the logic depth between t and n is no more than l, we update CPM by Eq. (5) (see Lines 9–12). Otherwise, we first obtain the l-th level boundary $S'_{n,l}$ of node n (see Line 14) and then update CPM by Eq. (6) (see Lines 15–19).

Algorithm 4: The function *GetCPM*(C, N_v, l) for computing the change propagation matrix P under the depth limit l.

```
Input  : a circuit $C$, a node value matrix $N_v$ obtained from
         MC simulation, and a depth limit $l$
Output : three-dimensional change propagation matrix $P$
1  $M \leftarrow$ number of samples in the MC simulation;
2  foreach PO $o$ in $C$ do
3      for $i$ from 1 to $M$ do
4          $P[i, o, o] \leftarrow 1$;
5          foreach sink node $n$ in $C$ except node $o$ do
6              $P[i, n, o] \leftarrow 0$;
7      foreach node $n$ in $C$ except node $o$ and sink nodes in
         reverse topological order do
8          get the one-cut $t$ between $n$ and $o$;
9          if logic depth between $t$ and $n \le l$ then
10             for $i$ from 1 to $M$ do
11                 compute $D[i, n, t]$ from matrix $N_v$;
12                 $P[i, n, o] \leftarrow P[i, t, o] \wedge D[i, n, t]$;
13         else
14             get the $l$-th level boundary $S'_{n,l}$ of node $n$;
15             for $i$ from 1 to $M$ do
16                 $P[i, n, o] \leftarrow 0$;
17                 foreach node $n_f$ in $S'_{n,l}$ do
18                     compute $D[i, n, n_f]$ from matrix $N_v$;
19                     $P[i, n, o] \leftarrow$
                         $P[i, n, o] \vee (P[i, n_f, o] \wedge D[i, n, n_f])$;
20 return $P$;
```

D. Batch Error Estimation

After we obtain the CPM, we can use it to do batch error estimation for all the ALTs of the current approximate circuit, as shown by Lines 6–7 in Algorithm 2. The key step is to obtain the error increase for an ALT based on the CPM. The flow of this step is shown in Algorithm 5. It is generally applicable to any statistical error measure. It takes an ALT A under consideration, the current approximate circuit C, and the CPM P for the circuit C as inputs. Besides, it also has the output value matrix O_{org} of the original circuit and the node value matrix N_{app} of the approximate circuit C as inputs. They are obtained from the MC simulation shown in Algorithm 2.

Line 4 extracts the output value matrix O_{app} of the circuit C from the matrix N_{app}, which will be used later. Line 5 identifies the local circuit C_L of C affected by the ALT A. Line 6 further gets the output node n_x of the local circuit C_L. Node n_x is an important node since it is the output interface to the remaining part of the circuit and the effect of the ALT can be observed from this node. In order to derive the error increase caused by the ALT, the procedure then obtains the MC simulation results of this node before and after we apply the ALT (see Lines 7–9). The MC simulation result of a node n is represented by a *signal value vector*, which is a size-M vector with the i-th entry giving the signal value of the node under the i-th input pattern. Line 7 obtains the signal value vector v_{app} of node n_x before we apply the ALT. Given that the node value matrix N_{app} stores the MC simulation results for all the nodes in the circuit C before we apply the ALT, the vector v_{app} can be simply extracted from N_{app}. Lines 8–9 derive the signal value vector v_{new} of node n_x after we apply the ALT. Specifically, Line 8 first applies the ALT A to the local circuit C_L to derive a new local circuit $C_{L,new}$. Line 9 then obtains the signal value vector v_{new}. Since the inputs of the local circuit $C_{L,new}$ are not affected by the ALT, thus, we only need to simulate $C_{L,new}$ from its local inputs using the input patterns stored in the node value matrix N_{app}. This avoids the costly re-simulating of the entire circuit.

Algorithm 5: The function *ErrorEstimate*(A, C, P, O_{org}, N_{app}) for estimating the error increase for an ALT.

```
1  Input: an ALT $A$, an approximate circuit $C$, a change propagation matrix $P$, and the output value matrix $O_{org}$ of the original circuit and the node value matrix $N_{app}$ of the approximate circuit $C$ obtained from the MC simulation;
2  Output: error increase $\Delta E$;
3  $\Delta E \leftarrow 0$; $M \leftarrow$ number of samples in the MC simulation;
4  $O_{app} \leftarrow$ the output value matrix of $C$ extracted from $N_{app}$;
5  $C_L \leftarrow$ the local circuit of $C$ affected by $A$;
6  $n_x \leftarrow$ the output node of $C_L$;
7  $v_{app} \leftarrow$ the signal value vector of node $n_x$ extracted from $N_{app}$;
8  apply $A$ to $C_L$ to get the new local circuit $C_{L,new}$;
9  simulate $C_{L,new}$ from its local inputs using the values in $N_{app}$ and obtain the signal value vector $v_{new}$ of node $n_x$ after we apply $A$;
10 for $i$ from $1$ to $M$ do
11     if $v_{app}[i] \neq v_{new}[i]$ then
12         $O_{new}[i,:] \leftarrow$ GetNewOutputs($O_{app}[i,:]$, $P[i,n_x,:]$);
13         $E_{app} \leftarrow$ Error($O_{app}[i,:]$, $O_{org}[i,:]$);
14         $E_{new} \leftarrow$ Error($O_{new}[i,:]$, $O_{org}[i,:]$);
15         $\Delta E \leftarrow \Delta E + E_{new} - E_{app}$;
16 return $\Delta E$
```

Then, the procedure goes through all the M input patterns and accumulates the error increase. For each input pattern, it first judges whether the value of n_x changes after we apply the ALT. This can be done by comparing $v_{app}[i]$ and $v_{new}[i]$ (see Line 11). If the value of n_x does not change, then for the current input pattern, all the PO values do not change after we apply the ALT. Consequently, the error increase is 0. Otherwise, Lines 12–15 calculate the error increase.

Assume that the errors of the new and the current approximate circuits are E_{new} and E_{app}, respectively, where the new approximate circuit is derived by applying the ALT to the current one. The error increase equals $E_{new} - E_{app}$. The notations $O_{new}[i,:]$, $O_{app}[i,:]$, and $O_{org}[i,:]$ represent the values of all the POs of the new approximate circuit, the current approximate circuit, and the original circuit, respectively, under the i-th input pattern. To calculate the error increase, Line 12 first gets the output values $O_{new}[i,:]$ of the new approximate circuit. They are obtained by the function *GetNewOutputs* based on the CPM and the output values $O_{app}[i,:]$ of the current approximate circuit. For any PO o, the function calculates $O_{new}[i,o]$ as follows:

$$O_{new}[i,o] = \begin{cases} \overline{O_{app}[i,o]}, & \text{if } P[i,n_x,o] = 1 \\ O_{app}[i,o], & \text{if } P[i,n_x,o] = 0 \end{cases}.$$

Line 13 calculates the value E_{app} by the function *Error* based on the output values of the current approximate circuit and those of the original circuit, while Line 14 calculates the value E_{new} by the same function based on the output values of the new approximate circuit and those of the original circuit. The function *Error*(A, B) calculates the error of the output values A of an approximate circuit over the correct output values B for one sample point in the MC simulation. The calculation depends on the error metric of interest and can be defined for any statistical error measure. We next illustrate how the function is implemented for two typical statistical error measures, ER and AEM.

- When the error measure is ER, the function returns 0 if the output values A and B are equivalent. Otherwise, it returns $\frac{1}{M}$, since we just consider 1 sample point out of M in the MC simulation.
- When the error measure is AEM, the function returns $\frac{1}{M}|\mathit{Bin}(A) - \mathit{Bin}(B)|$, where the function $\mathit{Bin}(X)$ gives the binary number encoded by X.

After E_{app} and E_{new} are obtained, the error increase $E_{new} - E_{app}$ for the current sample point is accumulated to the total error increase ΔE (see Line 15).

Since the error increase obtained by VECBEE is over the current approximate circuit, the amount of error increase may even be negative for some ALTs, which are favorable choices. Due to the high accuracy of VECBEE, we are able to identify these favorable ALTs and improve the quality of the synthesized approximate circuit.

E. Time Complexity Analysis

In this section, we analyze the time complexity of VECBEE for batch error estimation of all candidate ALTs in an iteration of the ALS flow. VECBEE is shown in Algorithm 2. Assume that the depth limit is l, the number of candidate ALTs is T, and the number of input patterns in the MC simulation is M. Assume that the circuit has N nodes, E edges, and O outputs. Thus, the average fanout number of the circuit is $b = E/N$. VECBEE involves two major steps: i) obtaining the CPM (see Line 4 in Algorithm 2) and ii) calculating the error increases for all the candidate ALTs (see Lines 6–7 in Algorithm 2).

For the first step, we analyze the general approach to compute the CPM, which is shown in Algorithm 4. It involves obtaining the one-cuts and calculating the CPM values by Eq. (5) or Eq. (6). The time complexity of obtaining the one-cuts is $\Theta(N(N+E)O)$. This is because we need to find the one-cut for each pair of node n and PO o and finding each one-cut requires a few invocations of the graph search algorithm and the topological sorting algorithm, which have time complexity $\Theta(N+E)$. To compute the CPM, we also need to compute the BD values by simulating a sub-circuit for each node n. The depth of the sub-circuit is roughly bounded by l. Given the average fanout number b, the size of each sub-circuit is $\Theta(b^l)$. Thus, the time complexity for simulating all the sub-circuits over all the input patterns is $\Theta\left(MNb^l\right)$. The time complexity of calculating the CPM by Eq. (6) is dominated by Line 19 in Algorithm 4. Since the size of the set $S'_{n,l}$ is $\Theta(b^l)$, the number of occurrences of Line 19 is $\Theta(MONb^l)$. In summary, the time complexity of Algorithm 4 is $\Theta\left(N(N+E)O + MNb^l(1+O)\right)$. Usually, $O \gg 1$, $M \gg N$, and $E = \Theta(N)$. Thus, the time complexity for computing the CPM is $\Theta\left(MNOb^l)\right)$.

To calculate the error increases for all the candidate ALTs, Algorithm 5 needs to be applied T times. The most time-consuming steps in the algorithm are the simulation of the local circuit $C_{L,new}$ at Line 9 and the loop from Line 10 to Line 15. The time complexity of the loop is $\Theta(MO)$, as several steps in the loop body work on vectors of size O. The time complexity of the local circuit simulation is $\Theta(M(N_L + E_L))$, where N_L and E_L are the number of nodes and the number of edges of the local circuit, respectively. Since the local circuit is typically small, the overall time complexity of Algorithm 5 is $\Theta(MO)$. Therefore, the total time complexity to obtain the error increases for all the candidate ALTs is $\Theta(MTO)$.

Thus, the time complexity of VECBEE is $\Theta\left(MO(Nb^l + T)\right)$. It should be noted that the number of candidate ALTs T depends on the specific ALS flow. For example, T is quadratic to N for SASIMI [22] and linear to N for ANS [23]. Nevertheless, we generally have $T = \Omega(N)$. For a small l, we can treat b^l as a constant. Consequently, the time complexity of VECBEE can be simplified to $\Theta\left(MOT\right)$. For comparison purpose, consider the traditional simulation-based method, which performs the MC simulation on the entire circuit for each candidate ALT to obtain its accurate error increase. In order to get the error increase for one ALT, the simulation runtime is $\Theta(M(N+E))$, where M is due to the M input patterns and $(N+E)$ is due to the forward propagation of the input values to the outputs. Given that there are T ALTs in total, the time complexity of the traditional simulation-based method is $\Theta(M(N+E)T)$. Since the number of outputs of a circuit is typically much smaller than its number of nodes, VECBEE is much more efficient than the traditional simulation-based method.

V. Experimental Results

In this section, we present the experimental results on VECBEE.

A. Experiment Setup

To study the effectiveness of VECBEE, we applied it to two existing greedy ALS flows, SASIMI [22] and ANS [23]. They are described in details in Section II. In each iteration, they evaluate the error increases and the quality improvement for all the ALTs and apply the one with the highest FOM calculated as the ratio of the quality improvement over the error increase. For these methods, they just perform quick but inaccurate error estimation.

Since we do not have the source code of SASIMI, we reimplemented it using C++. We used the logic synthesis tool SIS [42] 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 ALT does not increase the circuit delay. This gives the baseline SASIMI method we used in the experiments.

We applied VECBEE to both SASIMI and ANS. The resulting ALS flows are called *VECBEE-SASIMI* and *VECBEE-ANS*, respectively. For comparison purpose, we also realized a version of SASIMI enhanced with the traditional simulation-based method, in which we simulated the entire circuit for each ALT to evaluate its error. We call it *FULLSIM-SASIMI*.

All experiments were performed on a laptop with a quad-core 2.4GHz CPU (Intel I7-5500U) and a 8GB RAM. The number of random input patterns used in the MC simulation was 10000 for the experiments in Sections V-B and V-G, and 100000 for the others. We assumed that the PIs are uniformly distributed. The depth limit l used in Algorithm 4 is set to 1 unless otherwise specified. In the previous conference version of our work [35], we used the same set of random input patterns for all the iterations in the ALS flow. However, the approximate circuits generated in such a way may not satisfy the error threshold under another set of input patterns. Thus, in the following experiments, different sets of random input patterns were used in different iterations of the ALS flow. To avoid the influence of randomness, we performed the experiments in Sections V-E, V-F, and V-G 3 times and took the average value of area and delay, while the other experiments were performed only once.

The benchmark circuits used in our experiments are listed in Table I. They were well-optimized by SIS. The column "#nodes" lists the number of nodes in the Boolean network representation of a circuit. The column "#literals" lists the total number of literals in all node functions of the Boolean network. The circuits were mapped with the MCNC standard cell library [43].

TABLE I
Benchmark circuit information.

Name	I/O	Function	#nodes	#literals	Area	Delay
c880	60/26	8-bit ALU	357	633	599	40.4
c1908	33/25	16-bit SEC/DED circuit	880	759	1013	60.6
c2670	233/140	12-bit ALU and controller	1153	1357	1434	67.3
c3540	50/22	8-bit ALU	629	1674	1615	84.5
c5315	178/123	9-bit ALU	893	2461	2432	75.3
c7552	207/108	32-bit adder/comparator	1087	2552	2759	159.8
alu4	14/8	ALU	730	3199	2740	51.5
RCA32	64/33	32-bit ripple-carry adder	202	542	691	42.8
CLA32	64/33	32-bit carry-lookahead adder	303	843	1063	45.8
KSA32	64/33	32-bit kogge-stone adder	345	1031	1128	27.0
MUL8	16/16	8-bit array multiplier	436	978	1276	67.9
WTM8	16/16	8-bit wallace tree multipier	382	1008	1104	69.6

B. Accuracy of Monte Carlo Simulation

We validate the accuracy of Monte Carlo simulation in this experiment. Table II compares the simulated error obtained by MC simulation with the accurate error obtained by enumeration of all the input patterns. We considered both ER and

TABLE II

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

AEM. For each error measure, we considered two approximate circuits synthesized by SASIMI that are possible to enumerate all the input patterns. 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.

C. *Accuracy of VECBEE*

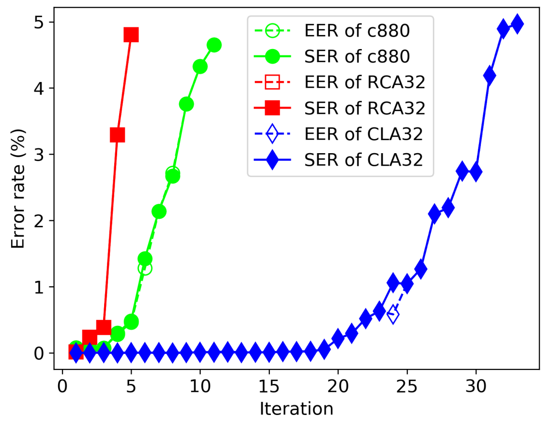

Fig. 4. The estimated ER (EER) by our method versus the simulated ER (SER) by the traditional simulation-based method on three benchmarks.

In this section, we studied the accuracy of VECBEE. We first compared the estimated ER (EER) by VECBEE with $l = 1$ and the simulated ER (SER) by the traditional simulation-based method for the approximate circuits of c880, RCA32, and CLA32 synthesized by SASIMI. The results are shown in Fig. 4. The horizontal axis of the figure gives the iteration number. As shown in Fig. 4, the EERs obtained by VECBEE is close to the SERs for all the iterations of the three circuits. However, due to the existing of reconvergent paths, EER and SER are not always equal, such as the 24th iteration of CLA32. We can also see that both the EER and the SER of CLA32 decrease in the 25th and 30th iteration. This demonstrates that VECBEE can effectively identify ALTs that can cause error decrease.

Then, we demonstrate that VECBEE can be configured to more accurate modes by changing the parameter l. We applied VECBEE with different values of l to estimate the ERs for six circuits c880, c1908, c2670, RCA32, CLA32, and KSA32. The values of l we chose are 1, 2, 4, and $+\infty$, where $l = +\infty$ corresponds to the fully accurate version of VECBEE. The golden ERs are obtained from the traditional simulation-based method. We used two metrics to evaluate the accuracy of VECBEE. The first is *correctness percentage (CP)*. It is defined as the percentage of the ALTs in the first three iterations of the ALS flow for which VECBEE obtains the golden ER. The second is *average error rate difference (AERD)*. It is defined as the average difference between the ER obtained by VECBEE and the golden one over all ALTs in the first three iterations of the ALS flow. The experimental results are shown in Table III. We can see that as the parameter l increases, CP increases monotonically, while AERD decreases monotonically. This shows that VECBEE becomes more accurate as l increases, which is expected. When l equals $+\infty$, CP becomes 100%, while AERD drops to 0. This verifies that VECBEE with $l = +\infty$ is fully accurate.

TABLE III

Accuracy of VECBEE under $l = 1,\ 2,\ 4,\ +\infty$ compared to the traditional simulation-based method. CP: correctness percentage; AERD: average error rate difference.

circuit	$l = 1$		$l = 2$		$l = 4$		$l = +\infty$	
	CP	AERD	CP	AERD	CP	AERD	CP	AERD
c880	62.84%	0.01350	74.76%	0.00550	84.62%	0.00190	100.00%	0.00000
c1908	52.21%	0.03850	86.30%	0.01280	88.98%	0.01200	100.00%	0.00000
c2670	58.46%	0.02330	86.54%	0.00570	87.37%	0.00540	100.00%	0.00000
rca32	92.10%	0.01590	99.45%	0.00010	99.90%	0.00001	100.00%	0.00000
cla32	42.34%	0.08150	95.13%	0.00320	98.42%	0.00003	100.00%	0.00000
ksa32	39.27%	0.10740	99.99%	0.00001	100.00%	0.00000	100.00%	0.00000

D. *Runtime and Synthesis Quality of VECBEE*

In this section, we studied the runtime and synthesis quality of VECBEE. We compared it with the traditional simulation-based method, which guarantees the accuracy of error estimation, but takes a long time. We applied both methods to the baseline SASIMI. This gives VECBEE-SASIMI and FULLSIM-SASIMI, as we mentioned in Section V-A. Since the traditional simulation-based method has a very long runtime, we only experimented on six small circuits, which are c880, c1908, c2670, RCA32, CLA32, and KSA32.

Fig. 5 shows the ratio of the runtime of VECBEE-SASIMI to that of FULLSIM-SASIMI. We considered 4 different choices of the parameter l for VECBEE as 1, 2, 4, and $+\infty$. From the figure, we can see that VECBEE-SASIMI is much faster than FULLSIM-SASIMI, even for VECBEE with $l = +\infty$. As expected, the runtime of VECBEE-SASIMI increases with l, but very slowly. On average, VECBEE-SASIMI with $l = 1$ achieves a speed-up of 85$\times$ over FULLSIM-SASIMI, while that with $l = +\infty$ achieves a speed-up of 78$\times$.

As for the synthesis quality, we compared the area ratios for different depth limits l under the ER constraint of 0.05%. The area ratio is the area of approximate circuit over that of the original accurate circuit. The result is shown in Fig. 6. As the depth limit l increases, the quality of the final approximate

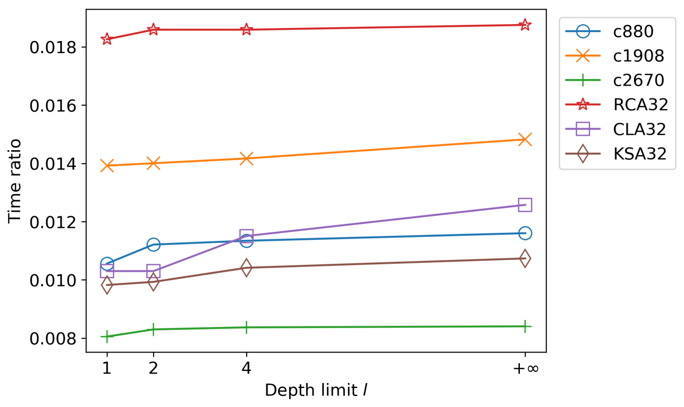

Fig. 5. The runtime ratio of VECBEE-SASIMI over FULLSIM-SASIMI for six benchmarks and different depth limits l of VECBEE.

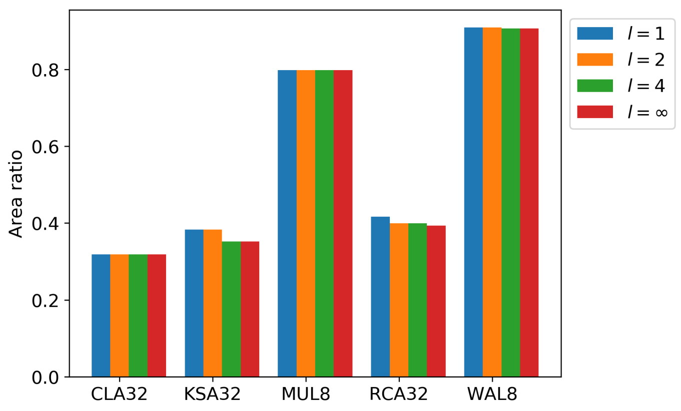

Fig. 7. Synthesis quality of VECBEE-SASIMI with different depth limits l under the AEM constraint of 0.00153%.

circuits keeps almost the same. This is reasonable, since VECBEE achieves high enough accuracy when $l = 1, 2, 4$ as shown in Table III and hence, the final synthesis quality is similar. We also compared the area ratios for different depth limits l under the AEM constraint of 0.00153%. The result is shown in Fig. 7. For the AEM constraint, we do see that as l increases, the areas of some approximate circuits generated by VECBEE-SASIMI decrease (e.g., KSA32 and RCA32), which indicates the need to improve the accuracy of the error estimation. We believe that the different behaviors between the ER constraint and the AEM constraint is due to the different sensitivities of the two error metrics to the accuracy of the error estimation.

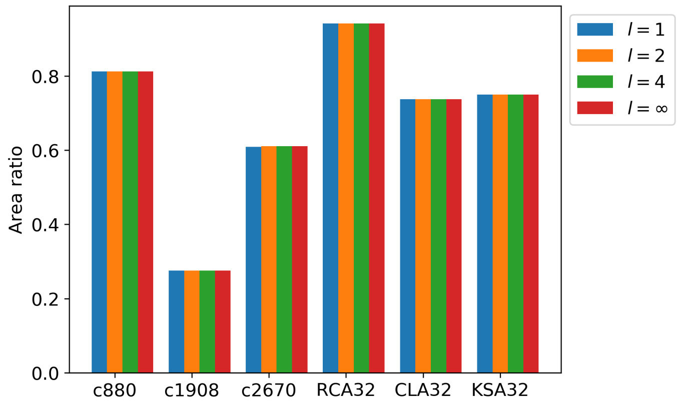

Fig. 6. Synthesis quality of VECBEE-SASIMI with different depth limits l under the error rate constraint of 0.5%.

E. Performance of VECBEE-SASIMI under Error Rate Constraint

In this section, we studied the performance of VECBEE-SASIMI under the ER constraint. We applied it to the benchmarks listed in Table I and measured the area and delay ratios of the approximate circuits over the original circuits. Fig. 8 plots how the area ratio changes with the ER. We can see that VECBEE-SASIMI can reduce 15%–35% area for most benchmarks under 5% ER threshold. We also compared it with the baseline SASIMI, which performs fast but inaccurate error estimation. The results are shown in Table IV, which lists the average area ratio and average delay ratio over seven ER thresholds (i.e., 0.1%, 0.3%, 0.5%, 0.8%, 1%, 3%, and 5%) for each benchmark. The entries in **bold** highlight the cases where VECBEE-SASIMI is better than the baseline SASIMI. It can be seen that for all the benchmarks, VECBEE-SASIMI gives smaller area than the baseline. For most benchmarks, the former also gives smaller delay than the latter. Especially, for the benchmark alu4, VECBEE-SASIMI can reduce 13.8% more area and 14.8% more delay than the baseline SASIMI. Considering all the benchmarks, VECBEE-SASIMI on average can reduce 4.4% more area and 4.5% more delay than the baseline. This demonstrates that VECBEE can truly improve the quality of SASIMI ALS flow under ER constraint.

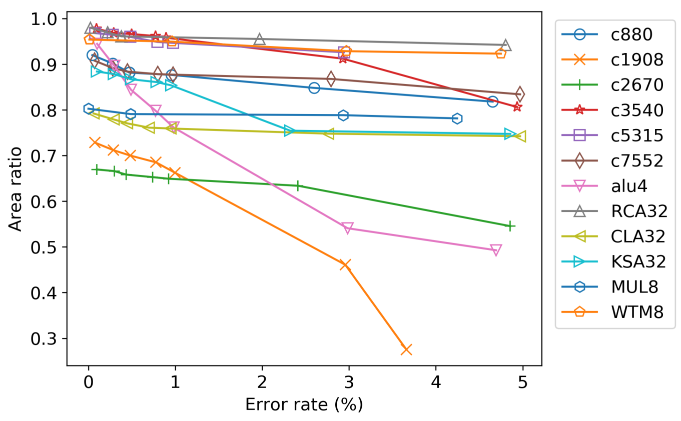

Fig. 8. Area ratios of the approximate circuits obtained by VECBEE-SASIMI over the original circuits for different ERs.

F. Performance of VECBEE-SASIMI under Average Error Magnitude Constraint

In this section, we studied the performance of VECBEE-SASIMI under the AEM constraint. We applied it to 5 arithmetic benchmarks listed in Table I, RCA32, CLA32, KSA32, MUL8, and WTM8. We measured the area ratios of the approximate circuits over the original circuits. Fig. 9

TABLE IV
COMPARISON BETWEEN THE BASELINE SASIMI AND VECBEE-SASIMI UNDER THE ER CONSTRAINT.

Circuit	Average area ratio		Average delay ratio	
	SASIMI	VECBEE-SASIMI	SASIMI	VECBEE-SASIMI
c880	0.896	**0.875**	0.953	**0.921**
c1908	0.610	**0.603**	0.965	**0.886**
c2670	0.724	**0.639**	0.682	**0.664**
c3540	0.975	**0.936**	0.991	**0.983**
c5315	0.981	**0.948**	0.989	0.991
c7552	0.948	**0.877**	0.977	0.979
alu4	0.892	**0.754**	0.969	**0.821**
RCA32	0.972	**0.961**	0.694	**0.652**
CLA32	0.829	**0.765**	1.251	**1.031**
KSA32	0.848	**0.835**	0.833	0.872
MUL8	0.829	**0.792**	0.925	**0.899**
WTM8	0.959	**0.945**	0.937	0.947
Arithmean	0.872	**0.828**	0.922	**0.887**

plots how the area ratio changes with the AEM. Its horizontal axis is the AEM rate, calculated as the AEM divided by the maximum binary number encoded by the outputs of a circuit. For AEM less than 0.2% of the maximum value, we can obtain 35%–85% area reduction.

We also compared VECBEE-SASIMI with the original SASIMI [22] for different AEM thresholds. The results for the original SASIMI were taken from [22] and we chose the same AEM thresholds as those in [22] for VECBEE-SASIMI. Table V lists the average area ratio over all the AEM thresholds for each benchmark. The entries in **bold** highlight the cases where VECBEE-SASIMI is better than the original SASIMI. For all the benchmarks, VECBEE-SASIMI saves much more area than the original SASIMI, although it does not even apply gate downsizing. On average, VECBEE-SASIMI has an improvement of 2.4$\times$ in area over the original SASIMI. The reason why the original SASIMI is much worse than VECBEE-SASIMI is because it only uses the signal probability difference between a pair of internal signals to guide the selection of an ALT and it cannot predict the errors at different POs. Therefore, it may choose ALTs that cause errors at the most significant bits, hence reaching the AEM threshold too quickly. This demonstrates that VECBEE is particularly helpful in synthesizing approximate circuits under the AEM constraint.

TABLE V
COMPARISON BETWEEN THE ORIGINAL SASIMI [22] AND VECBEE-SASIMI UNDER THE AEM CONSTRAINT.

Circuit	Average area ratio	
	SASIMI	VECBEE-SASIMI
RCA32	0.555	**0.193**
CLA32	0.423	**0.136**
KSA32	0.673	**0.140**
MUL8	0.626	**0.457**
WTM8	0.863	**0.390**
Arithmean	0.628	**0.263**

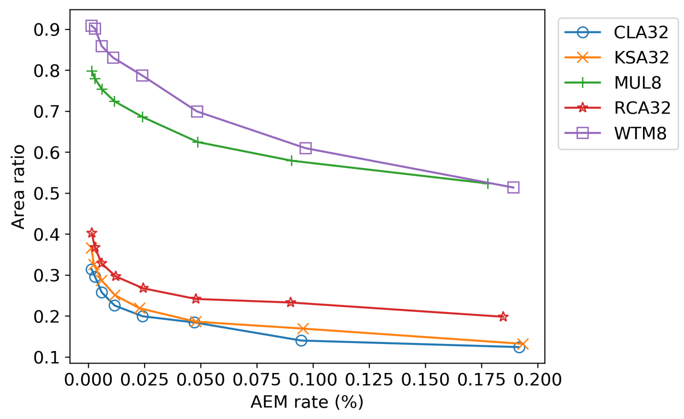

Fig. 9. Area ratios of the approximate circuits obtained by VECBEE-SASIMI over the original circuits for different AEMs.

TABLE VI
COMPARISON BETWEEN THE ORIGINAL ANS [23] AND VECBEE-ANS UNDER THE ER CONSTRAINT.

Circuit	Average literal ratio		Average area ratio		Average delay ratio	
	ANS	VECBEE-ANS	ANS	VECBEE-ANS	ANS	VECBEE-ANS
c880	0.904	**0.883**	0.891	**0.888**	0.942	**0.937**
c1908	0.830	0.864	0.536	0.567	0.685	0.746
c2670	0.797	**0.755**	0.635	**0.613**	0.627	**0.610**
c3540	0.978	**0.867**	0.989	**0.987**	0.933	**0.852**
c5315	0.982	**0.884**	0.938	0.950	0.661	**0.645**
c7552	0.972	**0.905**	0.911	**0.865**	0.734	**0.715**
alu4	0.875	**0.707**	0.758	0.761	0.665	0.680
RCA32	0.972	**0.957**	0.873	0.873	0.733	**0.727**
CLA32	0.913	**0.899**	0.738	**0.725**	0.831	0.836
KSA32	0.885	**0.845**	0.794	**0.778**	0.753	0.789
MUL8	0.982	**0.970**	0.806	0.811	0.829	0.841
WTM8	0.975	**0.936**	0.876	0.878	0.773	**0.758**
Arithmean	0.922	**0.873**	0.812	**0.808**	0.764	**0.761**

G. Performance of VECBEE-ANS under Error Rate Constraint

To further demonstrate the effectiveness and applicability of VECBEE, in this section, we applied it to the ANS ALS flow [23] and studied the resulting VECBEE-ANS ALS flow under the ER constraint. ANS works on the Boolean network representation of a circuit and optimizes the typical quality measure of a Boolean network, the literal count. Thus, we measured the total literal count in our experiment. Besides this, the areas and delays of the final circuits mapped from the Boolean networks were also obtained. We used ABC [44] to map the circuits and report the areas. Meanwhile, SIS was used for a more accurate delay measure, since SIS takes the effect of output load of gates into account when reporting the delay, while ABC does not.

The experimental results comparing the original ANS and VECBEE-ANS under the ER constraint are shown in Table VI. For each method, we obtained the literal count ratios, the area ratios, and the delay ratios of the approximate circuits over the original circuits. We considered 7 ER thresholds, which are 0.1%, 0.3%, 0.5%, 0.8%, 1%, 3%, and 5%, for each benchmark. The columns list the average literal count ratio, area ratio, and delay ratio over the seven ER thresholds for each benchmark. The entries in **bold** highlight the cases where VECBEE-ANS is better than the original ANS. For most benchmarks, VECBEE-ANS gives a smaller literal count than the original ANS. Considering all the benchmarks, VECBEE-ANS reduces 4.9% more literals and 0.4% more area than the original ANS on average. It is worth noting that the reduction in area is smaller than the reduction in literal count. We believe that this is because the technology mapping tool, which maps the Boolean network into the final circuit, does not always guarantee a strong correlation between the literal count and the circuit area. Nevertheless, since literal count is the major optimization target of ANS, the significant reduction in literal count still demonstrates the effectiveness of VECBEE.

VI. Conclusion

In this work, we proposed VECBEE, a versatile efficiency-accuracy configurable batch error estimation method for greedy ALS flows. Its key idea is to combine Monte Carlo simulation and local change propagation to estimate the influence of a batch of approximate local transformations on all the POs. It is generally applicable to any statistical error measure and graph representation of circuits. It allows runtime and accuracy trade-off and can be configured to a fully accurate version. The experimental results showed that VECBEE has very high error estimation accuracy. Furthermore, it is much faster than the traditional simulation-based error estimation method. We applied VECBEE to two existing ALS flows, SASIMI and ANS, and demonstrated that VECBEE can help improve the synthesis quality of these ALS flows.

References

[1] M. M. Waldrop, "The chips are down for Moore's law," *Nature*, vol. 530, no. 7589, pp. 144–147, 2016.

[2] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *European Test Symposium*, 2013, pp. 1–6.

[3] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design & Test*, vol. 33, no. 1, pp. 8–22, 2016.

[4] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys*, vol. 48, no. 4, 62:1–62:33, 2016.

[5] N. Zhu, W. L. Goh, and K. S. Yeo, "An enhanced low-power high-speed adder for error-tolerant application," in *Proceedings of the 12th International Symposium on Integrated Circuits*, 2009, pp. 69–72.

[6] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Design Automation Conference*, 2012, pp. 820–825.

[7] R. Ye, T. Wang, *et al.*, "On reconfiguration-oriented approximate adder design and its application," in *International Conference on Computer-Aided Design*, 2013, pp. 48–54.

[8] J. Hu and W. Qian, "A new approximate adder with low relative error and correct sign calculation," in *Design, Automation and Test in Europe*, 2015, pp. 1449–1454.

[9] V. Camus, M. Cacciotti, *et al.*, "Design of approximate circuits by fabrication of false timing paths: The carry cut-back adder," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 4, pp. 746–757, 2018.

[10] K. Y. Kyaw, W. L. Goh, and K. S. Yeo, "Low-power high-speed multiplier for error-tolerant application," in *International Conference of Electron Devices and Solid-State Circuits*, 2010, pp. 1–4.

[11] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *International Conference on VLSI Design*, 2011, pp. 346–351.

[12] C.-H. Lin and I.-C. Lin, "High accuracy approximate multiplier with error correction," in *International Conference on Computer Design*, 2013, pp. 33–38.

[13] C. Liu, J. Han, and F. Lombardi, "A low-power,high-performance approxiamte multiplier with configurable partial error recovery," in *Design, Automation and Test in Europe*, 2014, 95:1–95:4.

[14] S. Rehman, W. EI-Harouni, *et al.*, "Architectural-space exploration of approximate multipliers," in *International Conference on Computer-Aided Design*, 2016, 80:1–80:8.

[15] K. Nepal, Y. Li, *et al.*, "ABACUS: A technique for automated behavioral synthesis of approximate computing circuits," in *Design, Automation and Test in Europe*, 2014, 361:1–361:6.

[16] C. Li, W. Luo, *et al.*, "Joint precision optimization and high level synthesis for approximate computing," in *Design Automation Conference*, 2015, 104:1–104:6.

[17] S. Lee and A. Gerstlauer, "Data-dependent loop approximations for performance-quality driven high-level synthesis," *IEEE Embedded Systems Letters*, vol. 10, no. 1, pp. 18–21, 2018.

[18] J. Miao, A. Gerstlauer, and M. Orshansky, "Approximate logic synthesis under general error magnitude and frequency constraints," in *International Conference on Computer-Aided Design*, 2013, pp. 779–786.

[19] D. Shin and S. K. Gupta, "Approximate logic synthesis for error tolerant applications," in *Design, Automation and Test in Europe*, 2010, pp. 957–960.

[20] ——, "A new circuit simplification method for error tolerant applications," in *Design, Automation and Test in Europe*, 2011, pp. 1–6.

[21] S. Venkataramani, A. Sabne, *et al.*, "SALSA: Systematic logic synthesis of approximate circuits," in *Design Automation Conference*, 2012, pp. 796–801.

[22] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits," in *Design, Auntomation and Test in Europe*, 2013, pp. 1367–1372.

[23] Y. Wu and W. Qian, "An efficient method for multi-level approximate logic synthesis under error rate constraint," in *Design Automation Conference*, 2016, 128:1–128:6.

[24] A. Chandrasekharan, T. Villa, *et al.*, "Approximation-aware rewriting of AIGs for error tolerant applications," in *International Conference on Computer-Aided Design*, 2016, 83:1–83:8.

[25] Y. Wu, C. Shen, *et al.*, "Approximate logic synthesis for FPGA by wire removal and local function change," in *Asia and South Pacific Design Automation Conference*, 2017, pp. 163–169.

[26] Y. Yao, S. Huang, *et al.*, "Approximate disjoint bi-decomposition and its application to approximate logic synthesis," in *ICCD*, 2017, pp. 517–524.

[27] A. Ranjan, A. Raha, *et al.*, "ASLAN: Synthesis of approximate sequential circuits," in *Design, Automation and Test in Europe*, 2014, 364:1–364:6.

[28] J. Miao, A. Gerstlauer, and M. Orshansky, "Multi-level approximate logic synthesis under general error constraints," in *International Conference on Computer-Aided Design*, 2014, pp. 504–510.

[29] G. Liu and Z. Zhang, "Statistically certified approximate logic synthesis," in *ICCAD*, 2017, pp. 344–351.

[30] S. Froehlich, D. Groβe, and R. Drechsler, "Approximate hardware generation using symbolic computer algebra employing

grobner basis," in *Design, Automation and Test in Europe*, 2018, pp. 889–892.

[31] Z. Vasicek and L. Sekanina, "Evolutionary approach to approximate digital circuits design," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 432–444, 2015.

[32] Z. Zhou, Y. Yao, *et al.*, "DALS: Delay-driven approximate logic synthesis," in *International Conference on Computer-Aided Design*, 2019, pp. 1–7.

[33] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware aig rewriting: A fresh look at combinational logic synthesis," in *Design Automation Conference*, 2006, pp. 532–535.

[34] 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*, vol. 35, no. 5, pp. 806–819, 2016.

[35] S. Su, Y. Wu, and W. Qian, "Efficient batch statistical error estimation for iterative multi-level approximate logic synthesis," in *Design Automation Conference*, 2018, 54:1–54:6.

[36] S. Hashemi, H. Tann, and S. Reda, "Blasys: Approximate logic synthesis using boolean matrix factorization," in *Design Automation Conference*, 2018, pp. 55–60.

[37] I. Scarabottolo, G. Ansaloni, G. A. Constantinides, and L. Pozzi, "Partition and propagate: An error derivation algorithm for the design of approximate circuits," in *Design Automation Conference*, 2019, pp. 1–6.

[38] J. Echavarria, S. Wildermann, O. Keszocze, and J. Teich, "Probabilistic error propagation through approximated boolean networks," in *Design Automation Conference*, 2020, pp. 1–6.

[39] B. Krishnamurthy and I. G. Tollis, "Improved techniques for estimating signal probabilities," *IEEE Transactions on Computers*, vol. 38, no. 7, pp. 1041–1045, 1989.

[40] R. Ubar, J. Raik, *et al.*, "Multiple fault diagnosis with BDD based boolean differential equations," in *BBEC*, 2012, pp. 77–80.

[41] T. Y. Hsieh, K. J. Lee, and M. A. Breuer, "An error-oriented test methodology to improve yield with error-tolerance," in *VTS*, 2006, pp. 130–135.

[42] E. M. Sentovich, K. J. Singh, *et al.*, "SIS: A system for sequential circuit synthesis," University of California, Berkeley, Tech. Rep., 1992.

[43] S. Yang, "Logic synthesis and optimization benchmarks," Microelectronics Center of North Carolina, Tech. Rep., 1991.

[44] Berkeley Logic Synthesis and Verification Group, *ABC: A System for Sequential Synthesis and Verification, Release 80410*, Website, http://www.eecs.berkeley.edu/~alanmi/abc/, 2008.