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

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

***Abstract*—Approximate computing is an emerging 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 ALS. It is based on Monte Carlo simulation and an efficient technique to capture whether a signal change due to an introduced approximation will be propagated to each primary output. VECBEE is generally applicable to any statistical error measurement, such as error rate and average error magnitude, and any graph-based circuit representation. It 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 apply VECBEE to two representative greedy ALS methods and demonstrate 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 is error-tolerant applications, such as machine learning, data mining, and image processing. Given these trends, *approximate computing* [2]–[4] is 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]–[33].[1] In ALS, many studies target at the common circuit form, the multi-level circuit [20]–[33].

To explore the extremely large design space of an approximate circuit efficiently, many multi-level ALS methods [20]–[27] 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)* is then selected and applied.

For these methods, it is crucial 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* [21]. There are two groups of curves in the figure, where the upper and lower group shows 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%. The ERs of the circuits are derived by random simulation with 100000 samples. Since the randomness in the error estimation may affect the synthesis result, we repeat both SASIMI flows 80 times, and plot the

Sanbao Su (susannju@163.com) was and 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 Laboratory 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.

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

[1] We note that in a survey on ALS [34], approximate high-level synthesis (HLS) is classified as a sub-category of ALS. However, in our taxonomy, following the traditional view that HLS is not a sub-category of logic synthesis, we do not classify approximate HLS as a sub-category of ALS.

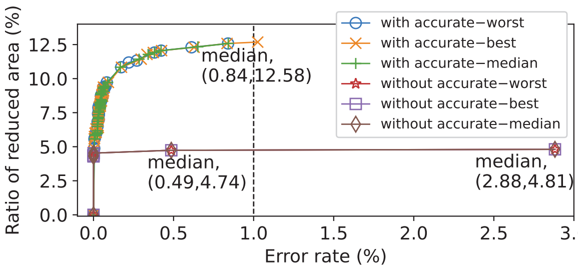


Fig. 1. Ratio of area reduction versus error rate for the same ALS method SASIMI with and without accurate error estimation.

curves that can achieve the best, the worst, and the median area reduction for each method, which correspond to the 3 curves in each group. Each point on a curve corresponds to the result after one round in the corresponding ALS flow. We can see that with 100000 samples, the 3 curves in each group are almost the same, but there is clear difference between the two groups of curves. The lower curves show that SASIMI without accurate error estimation stops after 3 rounds. It is because in the third round, due to the inaccurate error estimation, the best candidate ALT it selects actually increases the ER by 2.39%, 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 rounds and better quality for the final approximate circuit, as shown by the upper curves. For this example, the SASIMI method with accurate error estimation can reduce 7.84% more area than that without accurate error estimation in the median case.

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, it is time-consuming, since in order to get the errors for all the ALTs, its number of simulations equals the ALT number. 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 technique to capture whether a local signal value change due to an ALT will be propagated to each *primary output (PO)*. 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)* [35], *majority-inverter graph (MIG)* [36], 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 [21] and approximate node simplification (ANS) [22]. Our experiment results show that the ALS methods enhanced by VECBEE improve 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 [37], where we propose a basic batch error estimation method, which is efficient but not fully accurate. 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 the paper is organized as follows. Section II discusses the related works. Section III presents the preliminaries. Section IV elaborates the VECBEE methodology. Section V shows the experimental results. Finally, Section VI concludes the paper.

II. Related Works

This section discusses the related works on ALS for multi-level circuits and error estimation for approximate circuits.

A. ALS Methods for Multi-level Circuits

In this section, we briefly discuss some representative ALS methods for multi-level circuits. For a more general survey on ALS, the readers can refer to [34].

Several works propose greedy ALS methods for statistical error measure [20]–[24]. Shin and Gupta propose a greedy ALS method under ER and error magnitude constraint [20]. Its ALT is approximating a signal with a constant 0 or 1. ER is estimated by parallel random fault simulation. Venkataramani *et al.* introduce a greedy ALS method, SASIMI, which can handle either ER or AEM constraint [21]. 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. Wu and Qian propose a greedy ALS method under the ER constraint [22]. Its ALT is an approximate node simplification (ANS), which deletes some literals from the Boolean expression of a node in the network. The ER of each ALT is estimated as the ER at the output of each changed node, not the ER at the POs. The work [23] proposes 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 [22], it estimates ER from the output of the local circuit only. Hashemi *et al.* propose an ALS method with ALTs based on *Boolean matrix factorization (BMF)* [24]. Its ALTs are applied to sub-circuits that can have multiple outputs. For these works except [24], VECBEE can help improve their algorithm runtime or synthesis quality. For example, it can accelerate the error estimation and hence, the entire ALS flow of [20]. It can also improve the error estimation accuracy and hence, the synthesis quality for the works [21]–[23]. Since VECBEE can only handle ALTs affecting local circuits with a single output, it is not applicable to the ALS method in [24].

There are also ALS methods that are not for statistical error measure or not greedy. For example, the work [28] considers ALS under maximum error magnitude constraint. Since the targeted error metric is not a statistical error measure, VECBEE is not applicable. Another work proposes a non-greedy ALS flow [31]. In each iteration, it randomly selects an ALT and accepts it probabilistically. For early iterations, VECBEE may not help, as there is only one ALT under consideration and hence, an accurate error calculation for this single ALT is affordable. However, we can 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.

B. Error Estimation Methods for Approximate Circuits

Many prior works present error estimation methods for approximate circuits, which can be classified into module-level and gate-level methods.

The module-level methods mainly focus on circuits composed of approximate arithmetic modules, such as approximate adders and multipliers. They model and propagate the error through the approximate arithmetic modules [38]–[40]. For example, Sengupta *et al.* propose to obtain the probability mass function of approximate modules and propagate the error using the Fourier and the Mellin transforms [38]. Huang *et al.* propose to model and propagate errors of approximate modules using an interval-based approach [39].

The gate-level methods work on lower-level representation of the approximate circuits, such as partial product generators in multipliers or gates in gate netlists. They model and propagate the error at the gate level. They can be further divided into two categories based on the targeted circuit type.

The first category focuses on adders [41]–[43] and multipliers [44]. For example, Liu *et al.* propose a framework based on analytical models for evaluating the error characteristics of approximate adders [41]. Mazahir *et al.* present the error probability analysis for recursive approximate multipliers with approximate partial products [44].

The second category focuses on generic circuits, and our proposed method, VECBEE, belongs to it. For example, Venkatesan *et al.* propose to compute the error of approximate circuits using *satisfiability (SAT)* and *binary decision diagram (BDD)* [45]. However, their method is not scalable for large circuits, due to the high complexity of SAT and BDD. Scarabottolo *et al.* propose to partition the circuit into sub-circuits, and then determine the error propagation model of the resulting sub-circuits [46]. However, their method analyzes the maximum error magnitude, which is different from the statistical error metrics considered in our work. Echavarria *et al.* propose an error transition model that propagates the bit error rate through cascaded sub-circuits [47]. Instead of using logic simulation, it directly propagates the signal probabilities. Hence, it is faster than the simulation-based methods. Nevertheless, its accuracy is affected by reconvergent paths and the error magnitude metric is not supported.

We also highlight that different from most prior works on error estimation, VECBEE is an ALS-friendly approach, since it is designed to handle numerous approximate circuits considered in synthesis simultaneously. In that sense, the only comparable work to the best of our knowledge is [46], but it handles an error metric different from VECBEE.

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 m. 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 ER and 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

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

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

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

where $\left|\overrightarrow{f_{app}}(\vec{x}) - \overrightarrow{f_{org}}(\vec{x})\right|$ 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-A, 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-A.

In each iteration, the procedure chooses the optimal ALT and applies it to the current approximate circuit C_{app}. For this purpose, Line 3 first collects all possible ALTs of C_{app}. Then, Lines 5–6 evaluate the quality improvement ΔQ and the error increase ΔE caused by each ALT, where the quality can be area, delay, or power consumption and the error can 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

Algorithm 1: A general greedy ALS procedure.

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

FOM defined over ΔQ and ΔE is selected and applied (see Line 7). A typical FOM is $\Delta Q/\Delta E$, which favors an ALT that maximizes the quality improvement while minimizing the error increase. Finally, Line 8 calculates the actual error of the new approximate circuit. If it is smaller than the given error threshold E_{th}, the next iteration begins; otherwise, the entire loop finishes and the last approximate circuit satisfying the error bound, C_{out}, is returned.

IV. Methodology

This section elaborates the methodology of VECBEE.

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 [48] or BDD [49]. 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 [50].

From the general procedure shown in Algorithm 1, an important step is to evaluate the errors of all the 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 can lead to 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 [21]–[23]. 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, the 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 can be quite inaccurate. Below shows an example.

Example 1 *Consider the circuit shown in Fig. 2. Nodes $I_1, \ldots, I_7$ are the PIs and nodes O_1 and O_2 are the POs. Assume that the i-th input pattern in the MC simulation is $I_1 \ldots I_7 = 0111011$. The value of 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.*

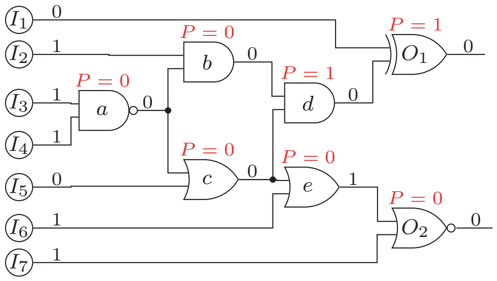


Fig. 2. An example circuit. The number above each wire is the signal value under the i-th input pattern in the MC simulation. The red number above each gate n is the change propagation matrix entry $P[i, n, O_1]$ computed by the method in Section IV-B, which is fast but sometimes inaccurate. Particularly, $P[i, a, O_1] = 0$ is inaccurate, and the accurate value $P[i, a, O_1] = 1$ can be obtained by the method in Section IV-C.

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 identify candidate ALT set $S$ from $C_{app}$;
5 $P \leftarrow$ GetCPM($C_{app}$, $N_{app}$);
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. Line 1 first generates M random input patterns, which are sampled from a given input distribution. Then, Lines 2–3 apply them to both the original circuit C_{org} and the current approximate circuit C_{app}, and produce 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 is 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 is 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.

Lines 5–7 are the key steps of VECBEE. 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 is a node in the circuit, and o is a PO of the circuit. The entry* $P[i, n, o] = 1$ *if and only if a value change on node n can be propagated to the output o under the i-th input pattern.*

Line 5 obtains the CPM for the current approximate circuit C_{app} based on the node value matrix N_{app} obtained before. 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 but sometimes inaccurate method to obtain CPM. Section IV-C extends the method to an efficiency-accuracy configurable version, which can be configured into a fully accurate version with a longer runtime. After obtaining the CPM, Line 7 estimates the error increase for each ALT of C_{app} based on the CPM and the matrices O_{org} and N_{app}. Its detail 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.

B. An Efficient Approximate Method to Obtain CPM

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)** *over 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, 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 $D[i, d, O_1] = 1$, $D[i, b, d] = 0$, $D[i, c, d] = 0$, $D[i, c, e] = 0$, $D[i, a, b] = 1$, and $D[i, a, c] = 1$.*

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 as its TFO, 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 satisfying that 1) the value change on n causes a value change on n_f and 2) the latter change can be propagated to the output o. These two conditions correspond to $D[i, n, n_f] = 1$ and $P[i, n_f, o] = 1$, respectively. Thus, we have the following recursive formula for calculating $P[i, n, o]$:

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

Example 3 *Consider the circuit in Fig. 2. Suppose that we want to get the CPM entries $P[i, n, O_1]$ for all the nodes n in Fig. 2 under 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*

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

From the above example we can see that by applying Eq. (4) *in a reverse topological order, the CPM entry $P[i, n, O_1]$ for each node n in Fig. 2 is obtained with a small amount of additional computation. This is clearly much faster than a straightforward method to obtain a CPM entry, in which the value of the corresponding node is first flipped and then the entire TFO of the node is re-simulated.*

From the above CPM entries, we can conclude that a value change on d can be propagated to O_1 for the i-th simulation *sample, while the changes on a, b, c, and e cannot. We also remark that all the above CPM entries except $P[i, a, O_1]$ are correct. The accurate value of $P[i, a, O_1]$ is actually 1. This inaccuracy will be analyzed in detail in Example 4.*

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.

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 is sometimes inaccurate due to the existence of reconvergent paths. Below 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 $a \to c \to d \to O_1$. Example 3 shows that $P[i, a, O_1] = 0$. 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 CPM Calculation: An Efficiency-Accuracy Configurable Approach

In this section, we show methods to improve the accuracy of CPM calculations. We first show how to obtain the fully accurate CPM. Then, we present a method to trade accuracy for runtime efficiency.

1) Calculating 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, $a \to b \to d \to O_1$ and $a \to c \to d \to O_1$. 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, and that 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 n and o divides the sub-circuit between n and o into two. 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 to 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. We get $d = 1$. 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 3, the new method 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 input node with an initial incoming flow of 1. The flow starts from the input 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 of 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: The last subsection shows a method to obtain the CPM accurately. However, it can take a long runtime. 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, 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 at most l, Eq. (5) is applied to compute an exact CPM entry. Otherwise, the following equation is used to compute the entry approximately:

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

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 that of a PO is labelled as $+\infty$. 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 number is the logic level of the corresponding node. Nodes o_1 and o_2 are POs and their logic levels are set to $+\infty$. 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 of level 3. Thus, the 3rd level boundary of a is $S'_{a,3} = \{e, o_2\}$.*

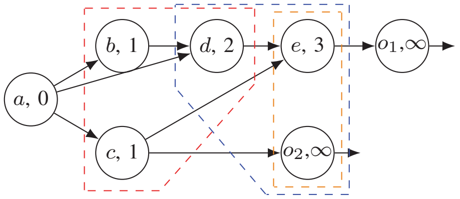


Fig. 3. Examples of l-th level boundary. The red, blue, and orange polygons circle out the 1st, 2nd, and 3rd level boundary of a, respectively.

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.

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 $P$;
$M \leftarrow$ number of samples in the MC simulation;
foreach PO $o$ in $C$ do
    for $i$ from 1 to $M$ do
        $P[i, o, o] \leftarrow 1$;
        foreach sink node $n$ in $C$ except node $o$ do
            $P[i, n, o] \leftarrow 0$;
    foreach node $n$ in $C$ except node $o$ and sink nodes in
      reverse topological order do
        get the one-cut $t$ between $n$ and $o$;
        if logic depth between $t$ and $n$ is at most $l$ then
            for $i$ from 1 to $M$ do
                compute $D[i, n, t]$ from matrix $N_v$;
                $P[i, n, o] \leftarrow P[i, t, o] \wedge D[i, n, t]$;
        else
            get the $l$-th level boundary $S'_{n,l}$ of node $n$;
            for $i$ from 1 to $M$ do
                $P[i, n, o] \leftarrow 0$;
                foreach node $n_f$ in $S'_{n,l}$ do
                    compute $D[i, n, n_f]$ from matrix $N_v$;
                    $P[i, n, o] \leftarrow$
                      $P[i, n, o] \vee (P[i, n_f, o] \wedge D[i, n, n_f])$;
return $P$;
```

Based on the above discussion, we can update Algorithm 3 into an efficiency-accuracy configurable version, which is shown in Algorithm 4. Besides the same inputs as Algorithm 3, it has an extra input, the depth limit l. 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 at most l, we obtain CPM by Eq. (5) (see Lines 9–12). Otherwise, Line 14 first obtains the l-th level boundary $S'_{n,l}$ of node n and then Lines 15–19 obtain CPM by Eq. (6).

We remark that Line 8 obtains the one-cut for each pair of node n and PO o, which is time-consuming for large circuits. We can accelerate this step by first analyzing the given circuit and recording whether for each pair of node n and PO o, the direct fanouts of n reconverge to o. If not, the corresponding CPM entry can be obtained by Eq. (4) without the need to compute the one-cut between n and o. Otherwise, we still get the one-cut between n and o, and then use Lines 9–19 in Algorithm 4 to get the CPM entry.

D. Error Estimation for a Single ALT

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. Its flow 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 2 extracts the output value matrix O_{app} of the circuit C from the matrix N_{app}, which will be used later. Line 3 identifies the local circuit C_L of C affected by the ALT A. Line 4 further gets the output node n_x of the local circuit C_L. Node n_x is important since the effect of the ALT can be observed from this node. In order to derive the error increase caused by the ALT, Lines 5–7 then obtain the MC simulation results of n_x before and after we apply the ALT. The 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. The signal value vector v_{app} of node n_x before we apply the ALT is extracted from the node value matrix N_{app} in Line 5. Lines 6–7 derive the signal value vector v_{new} of node n_x after we apply the ALT. Specifically, Line 6 first applies the ALT A to the local circuit C_L to derive a new local circuit $C_{L,new}$. Line 7 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.

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 9). 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 10–13 calculate the error increase.

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

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

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 10 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 11 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 12 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 $\frac{1}{M}|Bin(A) - Bin(B)|$, where the function $Bin(X)$ gives the binary number encoded by X.

Finally, Line 13 adds the error increase $(E_{new} - E_{app})$ for the current sample point to the total error increase ΔE.

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: 1) obtaining the CPM (see Line 5 in Algorithm 2) and 2) 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, $N = O(M)$ 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 7 and the loop from Line 8 to Line 13. 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 [21] and linear to N for ANS [22]. 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

This section presents the experimental results on VECBEE.

A. Experiment Setup

To study the effectiveness of VECBEE, we apply it to two existing greedy ALS flows, SASIMI [21] and ANS [22]. They are described in details in Section II-A. 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, as described in Section II-A.

Since we do not have the source code of SASIMI, we reimplement it using C++. We use the logic synthesis tool SIS [51] 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 do 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 use in the experiments.

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

All experiments are performed on a laptop with a quad-core 2.4GHz CPU (Intel I7-5500U) and a 8GB RAM. We assume 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 [37], we use 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 are used in different iterations of the ALS flow. To avoid the influence of randomness, we perform the experiments in Sections V-E, V-F, and V-G 3 times and report the averages of the metrics of interest, while the other experiments are performed only once.

In terms of the number of samples used in MC simulation, M, we find that setting it as 100000 generally gives a high-accuracy error estimation. Thus, we set $M = 100000$ unless otherwise specified. We also remark that if one looks for a more systematic way to customize M for different circuits, the method based on hypothesis test from [31] can be used.

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 detector/corrector	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 multiplier	382	1008	1104	69.6
MAC	32/17	multiplier & accumulator	560	1351	1372	57.5
EUDIST	32/16	Euclidean distance unit	1122	2813	2731	87.3
SAD	48/13	sum of absolute differences	425	962	999	70.9
square	64/128	64-bit square unit	14967	37843	37672	355.5
sqrt	128/64	128-bit square root unit	16584	42199	43859	7304
div	128/128	128-bit divisor	17710	44475	47469	5533.8
multiplier	128/128	128-bit multiplier	20260	54911	54205	419.5
log2	32/32	32-bit log2 unit	27468	72635	69688	651.4

Table I lists a wide range of circuits used in our experiments, including 6 ISCAS85 benchmark circuits [52], 9 arithmetic circuits, and 5 large EPFL benchmark circuits [53].[2] They are well-optimized by SIS. The column "#nodes" lists the number of nodes in the Boolean network representation of a circuit. Notably, the last 5 EPFL circuits all have more than 10000 nodes. The column "#literals" lists the total number of literals in all node functions of the Boolean network. The circuits are mapped with the MCNC standard cell library [54].

B. *Accuracy of Monte Carlo Simulation*

TABLE II
SIMULATED ER (SER) BY MC SIMULATION VERSUS ACCURATE ER (AER) AND SIMULATED AEM (SAEM) BY MC SIMULATION VERSUS ACCURATE AEM (AAEM). DIFFERENT ROWS CORRESPOND TO DIFFERENT APPROXIMATE VERSIONS OF A GIVEN CIRCUIT.

Approx. version	alu4		WTM8		MUL8		WTM8	
	SER(%)	AER(%)	SER(%)	AER(%)	SAEM	AAEM	SAEM	AAEM
ver. 1	0.390	0.361	0.210	0.244	1.751	1.750	1.863	1.875
ver. 2	0.550	0.549	0.250	0.244	3.784	3.750	3.795	3.797
ver. 3	0.870	0.885	0.570	0.629	7.390	7.437	7.780	8.008
ver. 4	1.120	1.068	1.090	1.010	13.729	13.758	15.589	15.621
ver. 5	3.070	3.033	2.890	2.923	25.574	29.939	29.028	28.839
ver. 6	5.190	5.060	4.310	4.388	59.0417	67.322	63.095	61.595

We validate the accuracy of MC simulation in this experiment. Table II compares the simulated error obtained by MC simulation with the accurate error obtained by enumerating all the input patterns. We consider both ER and AEM. For each error measure, we consider two approximate circuits synthesized by SASIMI that are possible to enumerate all the input patterns. Note that the minimum value of the total number of input patterns among these circuits is 2^{14} (i.e., given by alu4). For a meaningful study, the number of samples in the MC simulation, M, should be no more than 2^{14}. Therefore, we choose M as 10000 in this experiment instead of the default value 100000. Different rows in the table correspond to different approximate versions of a given circuit. 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*

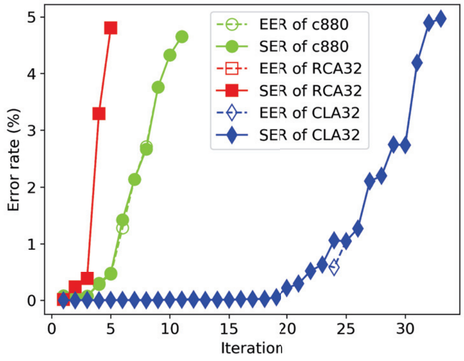


Fig. 4. The estimated ER (EER) by our method versus the simulated ER (SER) by the traditional simulation-based method on three benchmarks.

In this section, we study the accuracy of VECBEE. We first compare 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 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 3 circuits. However, due to the existence 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.

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	62.8	1.4E-02	74.8	5.5E-03	84.6	1.9E-03	100.0	0
c1908	52.2	3.9E-02	86.3	1.3E-02	89.0	1.2E-02	100.0	0
c2670	58.5	2.3E-02	86.5	5.7E-03	87.4	5.4E-03	100.0	0
RCA32	92.1	1.6E-02	99.5	1.0E-04	99.9	1.0E-05	100.0	0
CLA32	42.3	8.2E-02	95.1	3.2E-03	98.4	3.0E-05	100.0	0
KSA32	39.3	1.1E-01	100.0	1.0E-05	100.0	0	100.0	0
square	98.5	1.1E-04	99.0	6.3E-05	99.5	1.6E-05	100.0	0
sqrt	72.2	4.8E-03	72.7	4.7E-03	73.7	4.7E-03	100.0	0
div	100.0	4.4E-07	100.0	4.4E-07	100.0	6.2E-08	100.0	0
multiplier	97.2	1.5E-04	98.8	5.8E-05	100.0	0	100.0	0
log2	87.5	5.1E-04	87.5	5.0E-04	87.6	5.0E-04	100.0	0

Then, we demonstrate that VECBEE can be configured to more accurate modes by changing the parameter l. We test VECBEE-SASIMI with different values of l to estimate the ERs for the circuits listed in the first column of Table III. The

[2] We remark that some ISCAS85 benchmark circuits listed in Table I are inappropriate to be approximated, such as c1908, a 16-bit error detector/corrector. However, since they are used in SASIMI [21] and ANS [22], we also include them in our experiments for a comprehensive study.

values of l we choose are 1, 2, 4, and $+\infty$, where $l = +\infty$ corresponds to the fully accurate version of VECBEE. The golden ERs are obtained by the traditional simulation-based method. For the first 6 small circuits, we set M as the default value 100000, while for the last 5 large circuits from EPFL, M is set as 1000 to make the traditional simulation-based method feasible in runtime. We use two metrics to evaluate the accuracy of VECBEE. The first is *correctness percentage (CP)*. It is defined as the percentage of the ALTs in certain 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 certain iterations of the ALS flow. For the first 6 small circuits, we consider all the ALTs in the first 3 iterations of the ALS flow. For the last 5 large circuits, to make the traditional simulation-based method feasible in runtime, we only consider all the ALTs in the first iteration of the ALS flow.

The experimental results are listed in Table III. We can see that regardless of the circuit size, 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.

D. Runtime and Synthesis Quality of VECBEE

In this section, we study the runtime and synthesis quality of VECBEE. We compare it with the traditional simulation-based method, which guarantees the accuracy of error estimation, but takes a long time. We apply both methods to the baseline SASIMI. This gives VECBEE-SASIMI and FULLSIM-SASIMI, as we mentioned in Section V-A.

TABLE IV
RUNTIME OF FULLSIM-SASIMI AND VECBEE-SASIMI.

Circuit	Runtime of FULLSIM-SASIMI/s	Runtime of VECBEE-SASIMI/s			
		$l = 1$	$l = 2$	$l = 4$	$l = +\infty$
c880	767.1	8.1	8.6	8.7	8.9
c1908	1221.9	17.0	17.1	17.3	18.1
c2670	2843.6	22.9	23.6	23.8	23.9
RCA32	608.0	11.1	11.3	11.3	11.4
CLA32	1495.4	15.4	15.4	17.2	18.8
KSA32	1863.9	18.3	18.5	19.4	20.0
square	10297.6	111.3	113.7	120.0	155.7
sqrt	25604.8	203.6	207.0	221.1	711.0
div	46794.2	450.1	455.2	469.3	1588.2
multiplier	40213.9	476.2	477.3	491.4	645.3
log2	115161.6	911.1	920.8	931.0	1683.5
Arithmean	22442.9	204.1	206.2	211.9	444.1
Acceleration	1×	110×	109×	106×	51×

1) Runtime of VECBEE: Table IV compares the runtime of FULLSIM-SASIMI and VECBEE-SASIMI with different depth limit l's, using the same set of circuits listed in Table III. Similar to Section V-C, VECBEE-SASIMI and FULLSIM-SASIMI are applied for 3 iterations of the ALS flow with $M = 100000$ for the first 6 small circuits, and for one iteration of the ALS flow with $M = 1000$ for the last 5 large circuits.

From Table IV, we can see that regardless of the circuit size, VECBEE-SASIMI is much faster than FULLSIM-SASIMI, even for $l = +\infty$. As expected, the runtime of VECBEE-SASIMI increases with l. On average, VECBEE-SASIMI with $l = 1$ achieves a speed-up of $110\times$ over FULLSIM-SASIMI, while that with $l = +\infty$ achieves a speed-up of $51\times$.

From Table IV, we can also see that when $l \leq 4$, the runtime of VECBEE-SASIMI does not change a lot. However, when l changes from 4 to $+\infty$, the runtime of VECBEE-SASIMI shows a large increase for some large circuits (e.g., log2). The reason is that l only affects the runtime of getting the CPM. When $l \leq 4$, the runtime of getting the CPM does not increase much, and so is the overall runtime. When $l = +\infty$, the procedure of getting the CPM is the fully accurate version based on one-cut. Its runtime is dominated by the runtime of computing the BD value $D[i, n, t]$ as shown in Line 11 in Algorithm 4. For those large circuits with many long reconvergent paths, the runtime of computing all the BD values $D[i, n, t]$ through simulating each sub-circuit between a node n and its one-cut t increases significantly. Hence, the runtime of getting the CPM increases significantly, leading to a notable increase in the overall runtime.

TABLE V
RUNTIME OF VECBEE-ANS TO GENERATE APPROXIMATE EPFL CIRCUITS UNDER AN ER THRESHOLD OF 1% WITH $M = 100000$.

Circuit	square	sqrt	div	multiplier	log2
Runtime/h	8.23	20.85	25.81	18.84	11.83

We also remark that even accelerated by VECBEE, SASIMI takes an excessively long time to finish *all the rounds* of the ALS flow for the large EPFL circuits, since the number of ALTs given by SASIMI for a circuit is quadratic to the node number of the circuit [22]. However, for ALS methods with fewer number of ALTs, e.g., ANS, the version accelerated by VECBEE can handle the large circuits. Table V lists the runtime of VECBEE-ANS on the 5 large EPFL circuits under an ER threshold of 1% with the default value of M as 100000. The runtime ranges from 8.2 hours to 25.8 hours. The long runtime is mainly due to the large size of the circuits and hence, the large number of ALTs. Furthermore, the large number of ALTs also increases the number of iterations of the ALS flow significantly, as each iteration is more likely to identify an ALT with a very small induced error. Nevertheless, the data also highlights the usefulness of VECBEE, since if not accelerated by VECBEE, the runtime can reach more than 100 days for large circuits like div.[3]

2) Synthesis quality of VECBEE: As for the synthesis quality, we compare the area ratios for the depth limits $l = 1, 2, 4, +\infty$ under the AEM constraint of 0.00153% for the arithmetic circuits RCA32, CLA32, KSA32, MUL8, and WTM8. The area ratio is the area of the approximate circuit over that of the original circuit. The area ratios for different circuits and different l's are shown by the bars at the right of Fig. 5. We can see that as l increases, the quality of the final approximate circuits for some benchmarks (e.g., MUL8 and CLA32) 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. However, we do see that as l increases, the areas

[3] It is estimated based on the average acceleration ratio for $l = 1$ in Table IV.

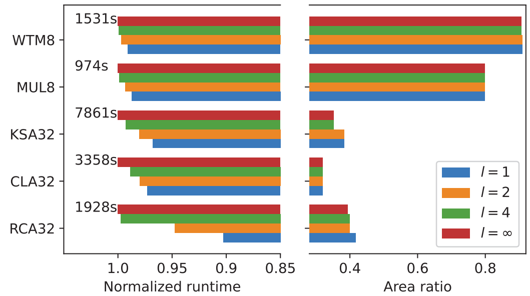


Fig. 5. Synthesis quality and runtime of VECBEE-SASIMI under the AEM constraint.

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.

To further understand the quality-runtime tradeoff, we also show the normalized runtime for different circuits with different l's by the bars at the left of Fig. 5. The number at the left of each red bar denotes the runtime for the corresponding circuit with $l = +\infty$, which is also the baseline value for normalization. As expected, the runtime increases with l. By considering the synthesis quality and runtime together, we can conclude that the quality improvement generally comes with a longer runtime, which is expected.

E. Performance of VECBEE-SASIMI under ER Constraint

This section studies the performance of VECBEE-SASIMI under the ER constraint. We apply it to a set of circuits listed in Table I, which are also used in [22], and measure the area and delay ratios of the approximate circuits over the original circuits. Fig. 6 plots how the area ratio changes with the ER. We can see that VECBEE-SASIMI can reduce 15%–35% area for most circuits under 5% ER threshold.

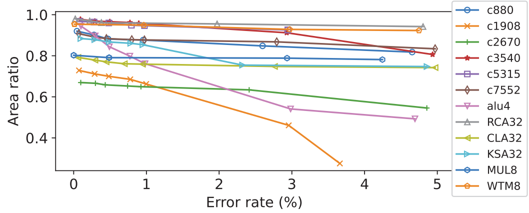


Fig. 6. Area ratios of the approximate circuits obtained by VECBEE-SASIMI over the original circuits for different ERs.

We also compare VECBEE-SASIMI with the baseline SASIMI, which performs fast but inaccurate error estimation. The results are shown in Table VI, which lists the average area ratio and average delay ratio over 7 ER thresholds (i.e., 0.1%, 0.3%, 0.5%, 0.8%, 1%, 3%, and 5%) for each circuit. The entries in **bold** highlight the cases where VECBEE-SASIMI is better than the baseline SASIMI. It can be seen that for all the circuits, VECBEE-SASIMI gives smaller area than the baseline. For most circuits, the former also gives smaller delay than the latter. Especially, for the circuit alu4, VECBEE-SASIMI can reduce 13.8% more area and 14.8% more delay than the baseline SASIMI. Considering all the circuits, 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. It is also noted that for the circuits c5135, c7552, KSA32, and WTM8, VECBEE-SASIMI increases the delay over the baseline SASIMI. The main reason is that VECBEE-SASIMI estimates the ALT errors more accurately than SASIMI. Hence, it selects a different and possibly better ALT in each iteration of the ALS flow than SASIMI. Since the ALTs always cause area reduction and less frequently cause delay reduction, these different choices always have a positive impact on area, but may induce a negative impact on delay.

TABLE VI
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**

F. Performance of VECBEE-SASIMI under AEM Constraint

In this section, we study the performance of VECBEE-SASIMI under the AEM constraint. We apply it to the last 8 arithmetic circuits listed in Table I, which are also used in [21]. Fig. 7 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 16%–85% area reduction.

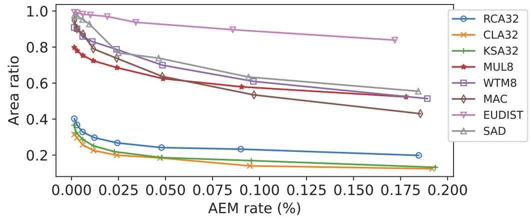


Fig. 7. Area ratios of the approximate circuits obtained by VECBEE-SASIMI over the original circuits for different AEMs.

We also compare VECBEE-SASIMI with the original SASIMI [21] for different AEM thresholds. The results for the original SASIMI are taken from [21], and we choose the same AEM thresholds as those in [21] for VECBEE-SASIMI. Table VII lists the average area ratio over all the AEM thresholds for each circuit. The entries in **bold** highlight the cases where VECBEE-SASIMI is better than the original SASIMI. For all the circuits, 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 $1.8\times$ in area over the original SASIMI. The reason why the original SASIMI is much worse than

VECBEE-SASIMI is that it only uses the signal probability difference between a pair of internal signals to guide the ALT selection 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 VII
COMPARISON BETWEEN THE ORIGINAL SASIMI [21] AND VECBEE-SASIMI UNDER THE AEM CONSTRAINT.

Circuit	Average area ratio		Circuit	Average area ratio	
	SASIMI	VECBEE-SASIMI		SASIMI	VECBEE-SASIMI
RCA32	0.555	**0.193**	WTM8	0.863	**0.390**
CLA32	0.673	**0.136**	MAC	0.794	**0.500**
KSA32	0.423	**0.140**	EUDIST	0.869	**0.785**
MUL8	0.626	**0.457**	SAD	0.657	**0.473**
Overall arithmean, SASIMI = 0.683, VECBEE-SASIMI=**0.384**					

G. Performance of VECBEE-ANS under ER Constraint

TABLE VIII
COMPARISON BETWEEN THE ORIGINAL ANS [22] 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**

To further demonstrate the effectiveness and applicability of VECBEE, in this section, we apply it to the ANS ALS flow [22] and study the resulting VECBEE-ANS ALS flow under the ER constraint. The same set of circuits listed in Table VI is used here. Note that we do not consider the AEM constraint, since ANS only targets at ER constraint. For a fair comparison to ANS, we set M as 10000, the same value used in ANS, here. 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 measure the total literal count here. Besides this, the areas and delays of the final circuits mapped from the Boolean networks are also obtained. We use ABC [55] to map the circuits and report the areas. Meanwhile, SIS is used for a more accurate delay measure, since SIS takes the 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 VIII. For each method, we obtain the literal count ratios, the area ratios, and the delay ratios of the approximate circuits over the original circuits. We consider 7 ER thresholds, which are 0.1%, 0.3%, 0.5%, 0.8%, 1%, 3%, and 5%, for each circuit. The columns list the average literal count ratio, area ratio, and delay ratio over the 7 ER thresholds for each circuit. The entries in **bold** highlight the cases where VECBEE-ANS is better than the original ANS. For most circuits, VECBEE-ANS gives a smaller literal count than the original ANS. Considering all the circuits, 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 propose VECBEE, a versatile efficiency-accuracy configurable batch error estimation method for greedy ALS flows. It is based our proposed CPM, which can be efficiently built to capture the influence of all the candidate ALTs 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 show that VECBEE has very high error estimation accuracy. Furthermore, it is much faster than the traditional simulation-based error estimation method. We apply VECBEE to two existing ALS flows, SASIMI and ANS, and demonstrate that VECBEE can help improve the synthesis quality of these ALS flows. Currently, VECBEE only supports ALTs affecting local circuits with a single output. In the future, we will study how to enhance VECBEE to support ALTs affecting local circuits with multiple outputs. A possible solution is to extend the definition of the current CPM.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their valuable comments. This work is supported by the National Key R&D Program of China Grant 2020YFB2205501, the State Key Laboratory of ASIC & System Open Research Grant 2019KF004, and the National Natural Science Foundation of China (NSFC) Grant 61574089.

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

[19] 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.

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

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

[22] 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.

[23] 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.

[24] S. Hashemi, H. Tann, and S. Reda, "BLASYS: Approximate logic synthesis using boolean matrix factorization," in *Design Automation Conference*, 2018, 55:1–55:6.

[25] A. Chandrasekharan, M. Soeken, D. Große, and R. Drechsler, "Approximation-aware rewriting of AIGs for error tolerant applications," in *International Conference on Computer-Aided Design*, 2016, 83:1–83:8.

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

[27] 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.

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

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

[30] 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.

[31] G. Liu and Z. Zhang, "Statistically certified approximate logic synthesis," in *International Conference on Computer-Aided Design*, 2017, pp. 344–351.

[32] 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.

[33] Z. Zhou, Y. Yao, *et al.*, "DALS: Delay-driven approximate logic synthesis," in *International Conference on Computer-Aided Design*, 2018, pp. 1–7.

[34] I. Scarabottolo, G. Ansaloni, G. A. Constantinides, L. Pozzi, and S. Reda, "Approximate logic synthesis: A survey," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2195–2213, 2020.

[35] 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.

[36] 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.

[37] 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.

[38] D. Sengupta, F. S. Snigdha, J. Hu, and S. S. Sapatnekar, "An analytical approach for error PMF characterization in approximate circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 1, pp. 70–83, 2019.

[39] J. Huang, J. Lach, and G. Robins, "A methodology for energy-quality tradeoff using imprecise hardware," in *Design Automation Conference*, 2012, pp. 504–509.

[40] W.-T. J. Chan, A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Statistical analysis and modeling for error composition in approximate computation circuits," in *International Conference on Computer Design*, 2013, pp. 47–53.

[41] C. Liu, J. Han, and F. Lombardi, "An analytical framework for evaluating the error characteristics of approximate adders," *IEEE Transactions on Computers*, vol. 64, no. 5, pp. 1268–1281, 2015.

[42] S. Mazahir, O. Hasan, R. Hafiz, M. Shafique, and J. Henkel, "Probabilistic error modeling for approximate adders," *IEEE Transactions on Computers*, vol. 66, no. 3, pp. 515–530, 2017.

[43] Y. Wu, Y. Li, X. Ge, Y. Gao, and W. Qian, "An efficient method for calculating the error statistics of block-based approximate adders," *IEEE Transactions on Computers*, vol. 68, no. 1, pp. 21–38, 2019.

[44] S. Mazahir, O. Hasan, R. Hafiz, and M. Shafique, "Probabilistic error analysis of approximate recursive multipliers," *IEEE Transactions on Computers*, vol. 66, no. 11, pp. 1982–1990, 2017.

[45] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "MACACO: Modeling and analysis of circuits for approximate computing," in *International Conference on Computer-Aided Design*, 2011, pp. 667–673.

[46] 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.

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

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

[49] R. Ubar, J. Raik, *et al.*, “Multiple fault diagnosis with BDD based boolean differential equations,” in *Biennial Baltic Electronics Conference*, 2012, pp. 77–80.

[50] T. Y. Hsieh, K. J. Lee, and M. A. Breuer, “An error-oriented test methodology to improve yield with error-tolerance,” in *VLSI Test Symposium*, 2006, pp. 130–135.

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

[52] M. Hansen *et al.*, “Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering,” *IEEE Design and Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.

[53] EPFL, *The epfl combinational benchmark suite*, https://lsi.epfl.ch/page-102566-en-html/benchmarks/, Accessed September 14, 2019.

[54] S. Yang, “Logic synthesis and optimization benchmarks,” Microelectronics Center of North Carolina, Tech. Rep., 1991.

[55] 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.

Sanbao Su received the B.S. degree in Automation from Nanjing University, China, in 2016, and the master’s degree in Electronic Science and Technology from the University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, China, in 2019. He is currently pursuing the Ph.D. degree in Computer Science and Engineering at the University of Connecticut, USA. His main research interests include reinforcement learning and optimal control.

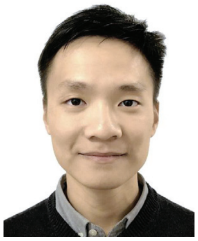
Chang Meng received the B.S. degree in Communication Engineering from Nanjing University of Science and Technology, China, in 2018. He is currently pursuing the Ph.D. degree in Electronic Science and Technology at the University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, China. His main research interests include logic synthesis and approximate computing.

Fan Yang (M’08) received the B.S. degree from Xi’an Jiaotong University in 2003, and the Ph.D. degree from Fudan University in 2008. From 2008 to 2017, he was an Assistant Professor and then an Associate Professor with Fudan University. He is currently a Professor with the Microelectronics Department, Fudan University. His research interests include model order reduction, circuit simulation, high-level synthesis, yield analysis, and design for manufacturability.

Xiaolong Shen is a senior engineer in Central Research Institute, Huawei Technologies Co., Ltd., Shenzhen, China. He received his Ph.D. degree in College of Computer Science and Technology at National University of Defense Technology. His main research interests include approximate computing, stochastic computing, computer architecture and high performance computing.

Leibin Ni (S’17-M’21) received the B.S. degree from Shanghai Jiao Tong University, China, and the Ph.D. degree from the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. He is currently a principal research engineer in Central Research Institute, Huawei Technologies Co., Ltd., Shenzhen, China. His main research interests include non-volatile memory, computing-in-memory, neuromorphic computing, asynchronous circuit, low-power design and cross-layer design.

Wei Wu is a hardware engineer at the Advanced Computing and Storage Laboratory of Huawei, Shenzhen. He received his Ph.D. degree in Electronic Science and Technology at Tsinghua University in 2019. His main research interests include emerging non-volatile memory and computing in memory.



Zhihang Wu received his master’s degree in optical engineering from Zhejiang University, China, in 2018. He then joined Huawei Technologies Co., Ltd. and have participated in several optical module development projects. His main research interests include high throughput and long distance optical communication system, digital signal processing algorithm and its low power implementation.

Junfeng Zhao is a technical expert of Huawei Technologies Co., Ltd. He has been working on data center technology for more than a decade. Over the past decade, he has focused on data center architecture technology research, as well as on advanced computing and advanced storage technologies related to data centers.

Weikang Qian (M’11-SM’21) is an associate professor in the University of Michigan-Shanghai Jiao Tong University Joint Institute at Shanghai Jiao Tong University. He received his Ph.D. degree in Electrical Engineering at the University of Minnesota in 2011 and his B.Eng. degree in Automation at Tsinghua University in 2006. His main research interests include electronic design automation and digital design for emerging computing paradigms. His research works were nominated for the Best Paper Awards at International Conference on Computer-Aided Design (ICCAD), Design, Automation, and Test in Europe Conference (DATE), and International Workshop on Logic and Synthesis (IWLS).