MECALS: A Maximum Error Checking Technique for Approximate Logic Synthesis

Chang Meng[1], Jiajun Sun[1], Yuqi Mai[1], and Weikang Qian[1,2]
[1]University of Michigan-SJTU Joint Institute and [2]MoE Key Lab of AI, Shanghai Jiao Tong University, Shanghai, China
Emails: {changmeng, sunjiajun2007, yq-mai, qianwk}@sjtu.edu.cn

***Abstract*—Approximate computing is an effective computing paradigm to improve energy efficiency for error-tolerant applications. Approximate logic synthesis (ALS) methods are designed to generate approximate circuits under certain error constraints. This paper focuses on ALS methods under the maximum error constraint and proposes MECALS, a maximum error checking technique for ALS. MECALS models maximum error using partial Boolean difference and performs fast error checking with SAT sweeping. Based on MECALS, we design an efficient ALS flow. Our experimental results show that compared to a state-of-the-art ALS method, our flow is $13\times$ faster and improves area and delay reduction by 39.2% and 26.0%, respectively.**

***Index Terms*—maximum error checking, approximate logic synthesis, partial Boolean difference, SAT sweeping**

I. Introduction

Approximate computing is an emerging low-power design paradigm for error-tolerant applications, such as image processing and machine learning [1]. *Approximate logic synthesis* (*ALS*) is an automatic way to generate approximate circuits [2]. An ALS tool takes an exact circuit and error constraints as inputs and explores the design space of approximate circuits with specific strategies. It produces an approximate circuit with a smaller area, power, and delay satisfying the constraints.

ALS typically handles two types of error metrics, *i.e.*, average and maximum errors [2]. Average errors, such as error rate and mean error distance, measure the average deviation between the outputs of exact and approximate circuits. Maximum errors, such as *worst-case error* (*WCE*) and *maximum square error* (*MaxSE*), compute the maximum deviation between the outputs of exact and approximate circuits over all input patterns. Many ALS methods are proposed for different error metrics [3]–[13]. The focus of our work is maximum error, which is essential in many scenarios. For example, in image processing, it is undesired that errors occur at the *most significant bits* (*MSBs*) of pixels; otherwise, an image will suffer from a large distortion. Another example is approximate arithmetic units, where MSBs are expected to be fully accurate. In these cases, the maximum error constraint can limit the errors only to the least significant bits. Several prior works can tackle the ALS problem under the maximum error constraint [8]–[13]. Among them, MUSCAT is a state-of-the-art (SOTA) method proposed recently [13]. Its basic approximate operation is replacing signals by constant 0s or 1s. Moreover, to find the optimal set of signals for replacement to maximize the area saving, a minimal unsatisfiable subset problem is formulated.

Most ALS methods simplify circuits by applying *local approximate changes* (*LACs*), *i.e.*, approximating local subcircuits [3]–[13]. To avoid violating the error constraint, we need to compute the maximum error of each LAC. A precise computation is impractical since it requires an exhaustive evaluation of all input patterns. To solve this issue, some works propose to estimate an upper bound of maximum error [9], [11], [12]. However, they are limited to a simple LAC that replaces a signal by a constant 0 or 1 [3]. More complex LACs such as signal substitution-based LACs [4], [7] that can further simplify a circuit are not supported. To facilitate the handling of complex LACs, another set of works directly checks whether the maximum error exceeds a bound or not [8], [10], [14]. They perform a one-by-one maximum error checking for each LAC by constructing an error miter and then solving the corresponding SAT problem. However, a new challenge they face is the massive number of complex LACs in a circuit, which makes checking them one by one very time-consuming.

To address the above challenge, in this work, we propose MECALS, an efficient maximum error checking technique for ALS, which supports complex LACs and significantly accelerates the maximum error checking of multiple LACs. Our main contributions are as follows:

- We establish an important theoretical foundation for checking the maximum error of a LAC based on *partial Boolean difference* (*PBD*). We develop both exact and approximate ways of computing PBDs.
- Based on the theoretical foundation, we design a *maximum error checking circuit* that can efficiently check the maximum error of *all* LACs using the *SAT-sweeping* tool.
- Based on the maximum error checking circuit, we develop an efficient ALS flow, *i.e.*, MECALS-based flow.

Our experimental results show that compared to a SOTA method, MUSCAT [13], our flow is $13\times$ faster and improves area and delay reduction by 39.2% and 26.0%, respectively. The code of our ALS flow is made open-source at https://github.com/SJTU-ECTL/MECALS.

II. Preliminaries

A. Partial Boolean Difference (PBD)

PBD is an important tool for diagnosing errors in circuits [15]. The 1st-order PBD of a single-output Boolean function $f = f(n_1, \ldots, n_i, \ldots n_M)$ *with respect to* (*w.r.t.*) one of its variables, n_i, is defined as

$$\Delta_{n_i} f = f(n_1, \ldots, n_i, \ldots, n_M) \oplus f(n_1, \ldots, \overline{n_i}, \ldots, n_M). \quad (1)$$

This work is supported by the National Key R&D Program of China under grant number 2021ZD0114701. Corresponding author: Weikang Qian.

By Eq. (1), if $f(n_1, \ldots, n_i, \ldots, n_M)$ and $f(n_1, \ldots, \overline{n_i}, \ldots, n_M)$ are different under a pattern on $n_1, \ldots, n_M$, then $\Delta_{n_i} f = 1$, implying that f changes after flipping the value of n_i in the pattern. Otherwise, f is not affected by the flip. When using Eq. (1) in a circuit, n_i can be any node, such as a *primary input* (*PI*), a *primary output* (*PO*), or an internal node, while f can be node n_i or any *transitive fanout* (*TFO*) of n_i.

Generally, the kth-order ($k \geq 2$) PBD of f *w.r.t.* k different variables $n_{i_1}, \ldots, n_{i_k}$ ($1 \leq i_1, \ldots, i_k \leq M$) is derived from the $(k-1)$th-order PBD as follows:

$$\Delta^k_{n_{i_1} \cdots n_{i_k}} f = \Delta_{n_{i_k}} \left(\Delta^{k-1}_{n_{i_1} \cdots n_{i_{k-1}}} f \right),$$

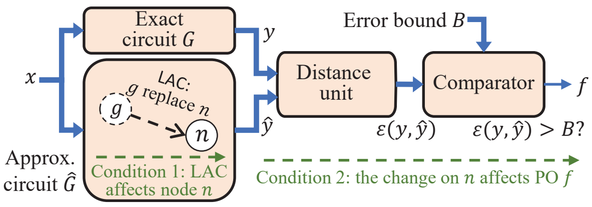
where $\Delta^1_{n_{i_1}} f = \Delta_{n_{i_1}} f$. For example, the 2nd-order PBD of f *w.r.t.* two variables n_i and n_j is $\Delta^2_{n_i n_j} f = \Delta_{n_j} (\Delta_{n_i} f) =$
$f(n_1,\ldots, n_i,\ldots, n_j,\ldots, n_M) \oplus f(n_1,\ldots, \overline{n_i},\ldots, n_j,\ldots, n_M) \oplus$
$f(n_1,\ldots, n_i,\ldots, \overline{n_j},\ldots, n_M) \oplus f(n_1,\ldots, \overline{n_i},\ldots, \overline{n_j},\ldots, n_M)$.
The kth-order PBD measures how f changes with the k variables $n_{i_1}, \ldots, n_{i_k}$. In what follows, for simplicity, PBD refers to the 1st-order PBD unless otherwise specified.

B. Maximum Error

Let $y : \mathbb{B}^I \to \mathbb{B}^O$ and $\hat{y} : \mathbb{B}^I \to \mathbb{B}^O$ be the multiple-output Boolean functions of an exact and an approximate circuit, respectively. The maximum error is defined as the maximum deviation between y and $\hat{y}$ over all possible input patterns:

$$\text{maximum error} = \max_{x \in \mathbb{B}^I} \varepsilon(y(x), \hat{y}(x)),$$

where $y(x)$ (*resp.* $\hat{y}(x)$) is the output of the exact (*resp.* approximate) circuit under the input pattern x, and ε is a distance function measuring the deviation between y and $\hat{y}$. For example, for WCE, $\varepsilon(y, \hat{y}) = |\mathit{int}(y) - \mathit{int}(\hat{y})|$, where the function $\mathit{int}(v)$ returns the integer encoded by the binary vector v. For MaxSE, $\varepsilon(y, \hat{y}) = [\mathit{int}(y) - \mathit{int}(\hat{y})]^2$.


Exact circuit G
x
y
Error bound B
LAC: g replace n
g
n
$\hat{y}$
Distance unit
Comparator
f
$\varepsilon(y,\hat{y})$
$\varepsilon(y,\hat{y}) > B$?
Approx. circuit $\widehat{G}$
Condition 1: LAC affects node n
Condition 2: the change on n affects PO f
Fig. 1. An error miter for maximum error check.

To check the maximum error, an *error miter* shown in Fig. 1 is proposed [14]. It consists of an exact circuit, an approximate circuit, a distance unit, and a comparator. The exact and approximate circuits take the same PIs x, and their corresponding outputs are y and $\hat{y}$, respectively. The distance unit computes $\varepsilon(y, \hat{y})$. The comparator checks whether $\varepsilon(y, \hat{y})$ exceeds the maximum error bound B. If the output of the comparator, f, is identically 0, *i.e.*, $f \equiv 0$, then the maximum error of $\widehat{G}$ does not exceed B; otherwise, the error constraint is violated. Such a check is usually done by converting the miter into a SAT instance solved by a SAT solver.

III. MECALS Methodology

This section presents the methodology of MECALS.

A. Motivation and Overview

Given an exact circuit G and a maximum error bound B, ALS generates an approximate circuit satisfying the bound. A widely-used ALS flow is the iterative ALS flow [2], which simplifies the circuit iteratively. For each iteration, there is a *current approximate circuit* $\widehat{G}$ satisfying the error constraint. To further simplify $\widehat{G}$, many candidate LACs are generated. However, some candidates may violate the given maximum error bound. Thus, the next step in the flow is to check whether *each* candidate LAC satisfies the bound, *i.e.*, the LAC is *valid*. Then, a valid LAC is selected and applied to simplify $\widehat{G}$.[1] As the number of LACs is typically large, *efficiently checking the validity of all the LACs* is challenging, and this is the problem MECALS tries to solve.

Note that our work only considers *single-output LAC* that replaces a single node n in $\widehat{G}$ with a new node g. For instance, a constant LAC [3] replaces n by $g = 0$ or $g = 1$. A SASIMI LAC [4] replaces n by $g = d$ or $g = \overline{d}$, where d can be any node in $\widehat{G}$ not depending on n. This type of LAC is the most widely-used one and can efficiently lead to approximate circuits of low cost, as reported in many prior works [3]–[5], [7], [8], [11], [13].

The following sections present our proposed solution, MECALS. First, Section III-B introduces a theorem on checking the validity of a LAC. Based on it, Section III-C presents the details of MECALS, namely, how to check the validity of all candidate LACs efficiently. MECALS builds upon a vital component: constructing a circuit for calculating various PBDs. Its detail is described in Section III-D.

B. Theorem on Checking the Validity of a LAC

To introduce the theorem, we start from a *current error miter* (*CEM*) V in the form shown in Fig. 1, which checks whether the current approximate circuit $\widehat{G}$ satisfies the maximum error bound. Initially, V's PO $f \equiv 0$, since $\widehat{G}$'s maximum error satisfies the bound (from $\widehat{G}$'s definition in Section III-A).

Now assume that $\widehat{G}$ is simplified by applying a LAC that replaces n by g. We denote the LAC as $C_{n,g}$. Checking the validity of the LAC $C_{n,g}$, *i.e.*, whether the error constraint is still satisfied after applying $C_{n,g}$, is equivalent to checking whether $f \equiv 0$ is still satisfied. Given that $f \equiv 0$ before applying $C_{n,g}$, it is equivalent to checking whether the function of f *remains unchanged* after applying $C_{n,g}$. By the miter structure shown in Fig. 1, if the function of f changes after applying $C_{n,g}$, then we can find an input pattern of x, say v, so that under pattern v, 1) the value of n changes after applying $C_{n,g}$, and 2) the change can be propagated to the PO f. Since $C_{n,g}$ means replacing n by g, the above condition 1 means that $n(v) \oplus g(v) = 1$, where $n(v)$ and $g(v)$ denote the value of n and g under pattern v. The above condition 2 means that $\Delta_n f(v) = 1$, where $\Delta_n f(v)$ denotes the value of the PBD $\Delta_n f$ under pattern v. Thus, if the function of f changes after applying $C_{n,g}$, we can find an input pattern letting $(n \oplus g)\Delta_n f$ yield 1. Otherwise, which means that the LAC $C_{n,g}$ is valid, we must have $(n \oplus g)\Delta_n f \equiv 0$. Thus, we conclude:

[1]For example, a simple yet efficient way to select a valid LAC is traversing each candidate LAC in a specific order and selecting the first valid one.

Theorem 1. *A LAC $C_{n,g}$ is valid* ***if and only if*** $(n \oplus g)\Delta_n f \equiv 0$.

C. Details of MECALS

This section elaborates MECALS, which checks the validity of all the LACs efficiently. It is based on Theorem 1. Assume that the nodes in the current approximate circuit $\widehat{G}$ are $n_1, n_2, \ldots, n_N$. For each node n_i $(1 \le i \le N)$, assume that there are L_i available single-output LACs, $C_{n_i,g_{i1}}, C_{n_i,g_{i2}}, \ldots, C_{n_i,g_{iL_i}}$, where the LAC $C_{n_i,g_{ij}}$ $(1 \le j \le L_i)$ replaces n_i by a new node g_{ij}.

By Theorem 1, checking the validity of $C_{n_i,g_{ij}}$ is equivalent to checking whether $(n_i \oplus g_{ij})\Delta_{n_i} f \equiv 0$. To efficiently check this condition for all the LACs $C_{n_i,g_{ij}}$'s, MECALS first builds a *maximum error checking circuit* shown in Fig. 2. It consists of a CEM V (in red), a *PBD circuit* (in green) that computes the PBDs $\Delta_{n_i} f$'s, and peripheral circuits to derive the *validity signal* $h_{ij} = (n_i \oplus g_{ij})\,\Delta_{n_i} f$ for each LAC $C_{n_i,g_{ij}}$. If $h_{ij} \equiv 0$, then the LAC $C_{n_i,g_{ij}}$ is valid; otherwise, it is invalid. The PBD circuit is an essential component in MECALS, which will be described in detail in Section III-D.

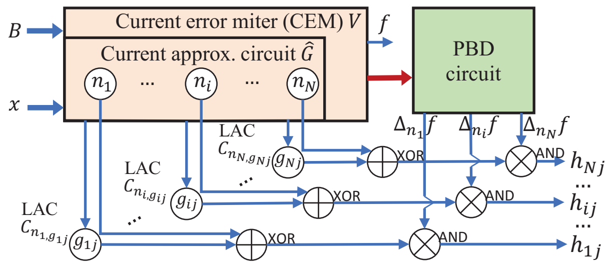


Fig. 2. Maximum error checking circuit. The signal h_{ij} is the validity signal for the LAC $C_{n_i,g_{ij}}$. The red arrow indicates that some internal signals of the CEM are passed to the PBD circuit.

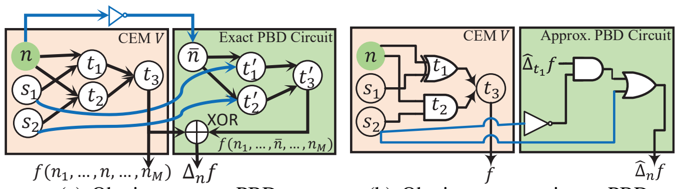
MECALS then applies SAT sweeping [16] to the maximum error checking circuit to check the validity of all LACs in $\widehat{G}$ efficiently. SAT sweeping is a powerful SAT-based method that detects functional equivalence in a circuit. Theoretically, after performing SAT sweeping on the maximum error checking circuit, we will find all h_{ij}'s that are identically 0 and the corresponding valid LACs. In reality, modern SAT-sweeping methods, such as the ABC [17] command "ifraig", are resource-limited for efficiency consideration. They can detect most functional equivalence cases in a circuit, while the remaining small portion may be unidentified. Therefore, using modern SAT sweeping, MECALS can also find most valid LACs whose h_{ij}'s are identically 0, while few valid LACs may remain unidentified. Compared to previous methods that check the validity of each LAC one by one [8], [10], [14], our method requires only one SAT sweeping on the maximum error checking circuit and then obtains the validity of all LACs, which is more efficient.

D. Construction of the PBD Circuit

This section describes how to construct the PBD circuit (the green box in Fig. 2), which computes the PBD $\Delta_{n_i} f$ for each node n_i in the current approximate circuit $\widehat{G}$. We propose exact and approximate ways of constructing the PBD circuit. For simplicity, we refer to n_i as n in this section.

1) Exact PBD Circuit: Fig. 3(a) shows the exact PBD circuit for calculating one $\Delta_n f$, which is based on the PBD definition shown in Eq. (1). The left part of Fig. 3(a) is the CEM, which directly gives $f(n_1, \ldots, n, \ldots, n_M)$. The right part computes the PBD $\Delta_n f$. Specifically, it first computes $f(n_1, \ldots, \overline{n}, \ldots, n_M)$ by copying n's TFOs (*i.e.*, t_1, t_2, and t_3) and driving the copied nodes (*i.e.*, t_1', t_2', and t_3') with $\overline{n}$. The new PO t_3' copied from the old PO t_3 gives $f(n_1, \ldots, \overline{n}, \ldots, n_M)$. Then, $\Delta_n f$ is obtained as the XOR of t_3 and t_3'. To compute all the PBDs $\Delta_n f$'s, the right part of Fig. 3(a) is repeated for each node n in the current approximate circuit.

However, if the CEM V and the current approximate circuit $\widehat{G}$ have M and N nodes, respectively, then $O(MN)$ nodes are copied to construct the PBD circuit for all the PBDs. As a result, for a large design, its maximum error checking circuit is too large to be handled by SAT sweeping. Thus, we need a more compact PBD circuit.



(a) Obtain an exact PBD. (b) Obtain an approximate PBD.

Fig. 3. Example circuits to compute PBDs.

2) Approximate PBD Circuit: To simplify the PBD circuit, we start from a recursive formula of computing PBDs exactly proposed in [15]. It calculates the PBD of n based on the PBDs of n's fanouts $u_1, u_2, \ldots, u_k$ $(k \ge 1)$ as follows:

$$\begin{aligned}
\Delta_n f = {} & (\Delta_n u_1 \cdot \Delta_{u_1} f) \oplus (\Delta_n u_2 \cdot \Delta_{u_2} f) \oplus \cdots \oplus (\Delta_n u_k \cdot \Delta_{u_k} f) \\
& \oplus (\Delta_n u_1 \cdot \Delta_n u_2 \cdot \Delta^2_{u_1 u_2} f) \oplus (\Delta_n u_1 \cdot \Delta_n u_3 \cdot \Delta^2_{u_1 u_3} f) \oplus \cdots \\
& \oplus (\Delta_n u_{k-2} \cdot \Delta_n u_k \cdot \Delta^2_{u_{k-2} u_k} f) \oplus (\Delta_n u_{k-1} \cdot \Delta_n u_k \cdot \Delta^2_{u_{k-1} u_k} f) \\
& \oplus \cdots \oplus (\Delta_n u_1 \cdot \Delta_n u_2 \cdots \Delta_n u_k \cdot \Delta^k_{u_1 u_2 \ldots u_k} f),
\end{aligned} \tag{2}$$

where $\Delta_n u_i$ $(1 \le i \le k)$ is the 1st-order PBD of the fanout u_i *w.r.t.* n, which can be obtained directly by Eq. (1). $\Delta_{u_i} f$ $(1 \le i \le k)$ is the 1st-order PBD of the output f *w.r.t.* the fanout u_i, $\Delta^2_{u_i u_j} f$ $(i \ne j, 1 \le i, j \le k)$ is the 2nd-order PBD of f *w.r.t.* u_i and u_j, and so on. The base case of the recursion is $\Delta_f f$, which equals 1 by definition. As Eq. (2) is too complex, we propose to approximate it. For simplicity, we illustrate our basic idea by the case where n has 2 fanouts, u_1 and u_2, which can be easily extended to general cases. In this case, Eq. (2) can be written as

$$\Delta_n f = (\Delta_n u_1 \cdot \Delta_{u_1} f) \oplus (\Delta_n u_2 \cdot \Delta_{u_2} f) \oplus (\Delta_n u_1 \cdot \Delta_n u_2 \cdot \Delta^2_{u_1 u_2} f) \tag{3}$$

Observing that the right-hand side (RHS) of Eq. (3) contains the terms $\Delta_n u_1$ and $\Delta_n u_2$, we can expand Eq. (3) by Shannon's expansion *w.r.t.* $\Delta_n u_1$ and $\Delta_n u_2$ as

$$\begin{aligned}
\Delta_n f = {} & \overline{\Delta_n u_1} \cdot \overline{\Delta_n u_2} \cdot \Delta_n f|_{\Delta_n u_1 = 0, \Delta_n u_2 = 0} + \Delta_n u_1 \cdot \overline{\Delta_n u_2} \cdot \Delta_n f|_{\Delta_n u_1 = 1, \Delta_n u_2 = 0} \\
& + \overline{\Delta_n u_1} \cdot \Delta_n u_2 \cdot \Delta_n f|_{\Delta_n u_1 = 0, \Delta_n u_2 = 1} + \Delta_n u_1 \cdot \Delta_n u_2 \cdot \Delta_n f|_{\Delta_n u_1 = 1, \Delta_n u_2 = 1},
\end{aligned} \tag{4}$$

where $\Delta_n f|_{\Delta_n u_1 = i, \Delta_n u_2 = j}$'s $(i, j \in \{0, 1\})$ are the *Shannon cofactors*.

To obtain the Shannon cofactor $\Delta_n f|_{\Delta_n u_1 = 0, \Delta_n u_2 = 0}$, we substitute $\Delta_n u_1$ and $\Delta_n u_2$ on the RHS of Eq. (3) with 0 and obtain it as 0. The other Shannon cofactors can be obtained similarly. Thus, Eq. (4) can be rewritten as

$$\Delta_n f = \Delta_n u_1 \cdot \overline{\Delta_n u_2} \cdot \Delta_{u_1} f + \overline{\Delta_n u_1} \cdot \Delta_n u_2 \cdot \Delta_{u_2} f + \Delta_n u_1 \cdot \Delta_n u_2 \cdot \left(\Delta_{u_1} f \oplus \Delta_{u_2} f \oplus \Delta^2_{u_1 u_2} f\right). \quad (5)$$

Since the 2nd-order PBD $\Delta^2_{u_1 u_2} f$ is not easy to derive, we propose to approximately set the Shannon cofactor containing the 2nd-order PBD, *i.e.*, $\Delta_{u_1} f \oplus \Delta_{u_2} f \oplus \Delta^2_{u_1 u_2} f$, as *one*. Then, an *approximate PBD* is computed as follows:[2]

$$\widehat{\Delta}_n f = \Delta_n u_1 \cdot \overline{\Delta_n u_2} \cdot \widehat{\Delta}_{u_1} f + \overline{\Delta_n u_1} \cdot \Delta_n u_2 \cdot \widehat{\Delta}_{u_2} f + \Delta_n u_1 \cdot \Delta_n u_2, \quad (6)$$

where $\widehat{\Delta}_n f$ is the approximate PBD of f *w.r.t.* n, and $\widehat{\Delta}_{u_1} f$ and $\widehat{\Delta}_{u_2} f$ are the approximate PBDs of f *w.r.t.* the fanouts u_1 and u_2, respectively. For the special case where $n = f$, we define $\widehat{\Delta}_f f = 1$.

By the definition of Eq. (6), an approximate PBD circuit can be constructed as follows. Initially, the node to compute $\widehat{\Delta}_f f$ is set as a constant 1. Then, each node n in the CEM V is traversed in a reverse topological order, and the corresponding node to compute $\widehat{\Delta}_n f$ is added according to Eq. (6). Fig. 3(b) gives an example of building a node to compute $\widehat{\Delta}_n f$. Assume n has two fanouts, t_1 and t_2, whose functions are $t_1 = n \oplus s_1$ and $t_2 = n s_2$, respectively. By Eq. (1), we have $\Delta_n t_1 = 1$ and $\Delta_n t_2 = s_2$. Then, by Eq. (6), we further have

$$\widehat{\Delta}_n f = \Delta_n t_1 \overline{\Delta_n t_2} \widehat{\Delta}_{t_1} f + \overline{\Delta_n t_1} \Delta_n t_2 \widehat{\Delta}_{t_2} f + \Delta_n t_1 \Delta_n t_2 = \overline{s_2} \widehat{\Delta}_{t_1} f + s_2,$$

and its actual implementation is shown in the right part of Fig. 3(b). Compared to the exact PBD circuit, the approximate PBD circuit does not need to copy the TFOs of a node n. Instead, $\widehat{\Delta}_n f$ only depends on the approximate PBDs of n's fanouts. Therefore, an approximate PBD circuit is much smaller and more practical than an exact PBD circuit.

Although $\widehat{\Delta}_n f$ does not equal $\Delta_n f$, it can be proved that if $\widehat{\Delta}_n f = 0$, then $\Delta_n f = 0$, *i.e.*, $\widehat{\Delta}_n f = 0$ implies $\Delta_n f = 0$. By this property, we further deduce the following theorem.

Theorem 2. *A LAC $C_{n,g}$ is valid **if** $(n \oplus g)\widehat{\Delta}_n f \equiv 0$.*

Theorem 2 can be easily proved by discussing two possible cases: 1) $n \oplus g = 0$ and 2) $\widehat{\Delta}_n f = 0$. Its detailed proof is omitted. Based on Theorem 2, we can *approximately* check the validity of each LAC by checking whether $(n \oplus g)\widehat{\Delta}_n f \equiv 0$. If it is true, then the LAC is valid. Otherwise, the LAC's validity is unknown, and conservatively, we treat the LAC as invalid.

3) Circuit Combining Exact and Approximate PBDs: Although an approximate PBD circuit is smaller than an exact PBD circuit, it is also less accurate. To improve the accuracy, we further propose a circuit combining exact and approximate PBD computation. In this circuit, exact and approximate PBD computation is applied to nodes with more and fewer fanouts, respectively. The reason is that a node with more fanouts would have more terms containing non-first-order PBDs discarded during the approximation from Eq. (5) to Eq. (6). Thus, the approximate PBD for a node with many fanouts suffers from a significant error compared to the exact PBD. Therefore, for nodes with more fanouts, exact PBDs are computed to guarantee accuracy, while for nodes with fewer fanouts, approximate PBDs are used for efficiency.

In practice, we decide the PBD computation mode for each node in the current approximate circuit $\widehat{G}$ as follows. For each output of $\widehat{G}$, we compute the exact PBD. For each internal (*i.e.*, non-input and non-output) node, its PBD computation mode is decided by a parameter $P \in [0, 100]$. Specifically, the top $P\%$ internal nodes with the most fanouts use exact PBDs, while the rest $(100 - P)\%$ use approximate PBDs. With the PBD computation mode decided, the PBD circuit for each internal node of $\widehat{G}$ is constructed in a reverse topological order.

IV. ALS Flow Based on MECALS

This section presents an ALS flow based on MECALS, shown in Algorithm 1. The inputs are an exact circuit G, a maximum error bound B, and a percentage of nodes using exact PBDs, P. Its output is an approximate circuit $\widehat{G}$ that satisfies the error constraint. The flow initializes the current approximate circuit $\widehat{G}$ with the exact circuit G (Line 1) and then approximates the circuit iteratively (Lines 2–11).

Algorithm 1: MECALS-based ALS procedure.

```
Input: exact circuit $G$, maximum error bound $B$, percentage
       of nodes using exact PBDs, $P$;
Output: approximate circuit $\widehat{G}$;
1  current approximate circuit $\widehat{G} \leftarrow G$;
2  while true do
3      $V \leftarrow$ BuildCurrentErrorMiter($G, \widehat{G}, B$);
4      $G_p \leftarrow$ BuildPBDCircuit($\widehat{G}, V, P$);
5      Obtain the set of candidate LACs $\mathbb{C}$ in $\widehat{G}$;
6      $G_e \leftarrow$ BuildErrorCheckCircuit($V, G_p, \mathbb{C}$);
7      SATSweeping($G_e$);
8      foreach LAC $C_{n_i,g_{ij}} \in \mathbb{C}$ do
9          $h_{ij} \leftarrow$ validity signal of $C_{n_i,g_{ij}}$ in $G_e$;
10         if $h_{ij} \equiv 0$ then ApplyLAC($\widehat{G}, C_{n_i,g_{ij}}$); break;
11     if $\widehat{G}$ is not modified then break;
12 return $\widehat{G}$;
```

In each iteration, Line 3 builds a CEM V to check the maximum error of $\widehat{G}$. Based on the circuit V, Line 4 further constructs a PBD circuit G_p that combines exact and approximate PBDs by the method described in Section III-D3. Then, Line 5 obtains the set of candidate LACs $\mathbb{C}$ in $\widehat{G}$. Two kinds of LACs, the constant LAC [3] and the SASIMI LAC [4], are considered in our implementation. Next, Line 6 builds a maximum error checking circuit G_e as shown in Fig. 2, and Line 7 performs SAT sweeping on G_e. After that, Line 8 visits each LAC $C_{n_i,g_{ij}}$, and Line 9 obtains the corresponding validity signal h_{ij} from G_e. Once a valid LAC $C_{n_i,g_{ij}}$ is met, *i.e.*, the corresponding h_{ij} is identically 0, Line 10 applies the LAC to $\widehat{G}$ and skips the rest LACs. Then, the flow continues with the next iteration. The flow terminates until the approximate circuit $\widehat{G}$ is not modified, which means that there are no valid LACs (Line 11). Finally, Line 12 returns the approximate circuit $\widehat{G}$.

It is worth mentioning that the order of evaluating candidate LACs (Lines 8–10) affects the synthesis quality. In our implementation, the candidate LACs are sorted by two keys. The primary key is the LAC type: the constant LACs are evaluated

[2]In the general case where n has k ($k \geq 1$) fanouts, we similarly set all the Shannon cofactors containing non-first-order PBD terms as *one* to get the approximate PBD.

before the SASIMI LACs. The reason is that by replacing a node n by 0 or 1, we can perform constant propagation to reduce circuit area significantly. The secondary key is the logic level of the node that a candidate LAC approximates. In other words, if a candidate LAC approximates a node with a smaller logic level, then it is considered first. It is because a node with a smaller logic level has a larger TFO cone, and approximating it may bring more area saving.

V. Experimental Results

In this section, we present the experimental results. We implement MECALS and the MECALS-based ALS flow in C++ and test them on a single core of an AMD 4800H processor with 32GB RAM. Currently, our flow works on an AND-inverter graph-based representation of circuit, although it also supports other circuit representations. Our flow integrates ABC [17], a SOTA logic synthesis and verification system. It utilizes various ABC commands, including "amap" for mapping circuits with the Nangate 45nm library [18], "stime" for measuring the area and delay of mapped circuits, and "ifraig" for SAT sweeping. Tested benchmarks are listed in Table I, whose columns show circuit names, PI/PO/gate numbers, area, and delay. They are exact circuits, including those used in MUSCAT [13], a SOTA method proposed recently, some BACS benchmarks [19], adders, and multipliers.

TABLE I. Benchmark circuits. Area (*resp.* Delay) is in μm^2 (*resp.* *ps*). "*": used in MUSCAT [13]; "+": from BACS [19].

Name	#I/#O/#Gate	Area	Delay	Name	#I/#O/#Gate	Area	Delay
absdiff*$^+$	16/8/80	87.3	416.3	buttfly$^+$	32/34/187	170.5	1008.2
add8*	16/9/41	42	359.9	mac$^+$	12/8/91	92.8	595.9
add32*$^+$	64/33/190	184.6	1843.3	mult8*	16/16/422	435.4	1255.2
add128	256/129/907	933.4	960.2	mult16$^+$	32/32/1528	1418.8	1981.5
binsqrd*	16/18/1047	1052.3	1526.8	mult32	64/64/5819	5723.3	1873.5

A. Accuracy-Efficiency Tradeoff of MECALS

This section studies the accuracy-efficiency tradeoff of MECALS. We compare it with a SAT-based exact maximum error checking method (called *exact method* in short), which is used in many previous works [8], [10], [14]. The exact method checks the validity of LACs one by one. It applies each LAC into the current approximate circuit $\widehat{G}$ and obtains a new approximate circuit G_{new}. Then, an error miter for G_{new} is constructed and converted into a SAT instance. After that, a SAT solver is called to check the validity.

Notably, the maximum error checking problem is a binary classification problem that divides candidate LACs in $\widehat{G}$ into valid and invalid LACs. Therefore, the accuracy of MECALS can be evaluated by two standard metrics for binary classifiers, *precision* and *recall*, defined as follows:

$$\text{precision} = TP/(TP + FP), \ \text{recall} = TP/(TP + FN),$$

where *TP* (*true positive*) is the number of LACs checked as valid by both MECALS and the exact method, *FP* (*false positive*) is the number of LACs checked as valid by MECALS but invalid by the exact method, and *FN* (*false negative*) is the number of LACs checked as invalid by MECALS but valid by the exact method. We prefer higher precision and recall, and as a perfect classifier, the precision and recall of the exact method are both 1.

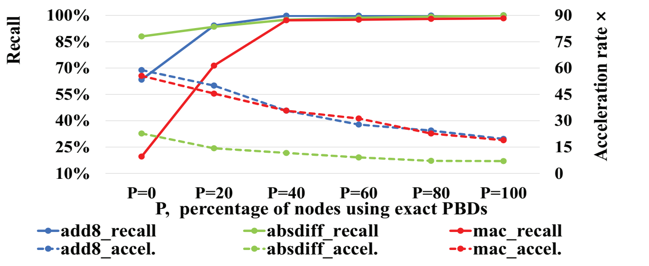


Fig. 4. Recall and acceleration rate versus P, the percentage of nodes using exact PBDs. The WCE bound is 16.

We compare MECALS with the exact method on three benchmarks, *absdiff*, *add8*, and *mac*, under a WCE bound of 16. Only the first three iterations of our proposed ALS flow (Algorithm 1) are considered, and *TP*, *FP*, and *FN* are counted in these iterations.

The first observation is that *FP* is always 0, and the precision is always 1. Thus, no LACs are checked as valid by MECALS but invalid by the exact method. This observation is consistent with Theorem 2, which guarantees that each valid LAC found by MECALS can be applied to simplify the circuit without violating the error constraint.

Then, we analyze the relationship between recall of MECALS and P, the percentage of nodes using exact PBDs (see Section III-D3). Fig. 4 plots recall versus P. We can see that when $P \geq 40$, the recall for each circuit is almost 100%, implying that MECALS can detect almost the same set of valid LACs as the exact method. Furthermore, as P increases, recall increases for all benchmarks and approaches 1. However, for circuit *mac*, when $P = 100$ (*i.e.*, all nodes use exact PBDs), recall is not exactly 1. This implies that $FN \neq 0$, *i.e.*, some LACs are found as invalid by MECALS but valid by the exact method. In other words, even using exact PBDs ($P = 100$), MECALS cannot detect all valid LACs. This is because the ABC command "ifraig", as a resource-limited SAT-sweeping method, does not detect all functional equivalence cases in the maximum error checking circuit (see Section III-C).

Fig. 4 also plots the acceleration rate, defined as the runtime of the exact method over that of MECALS, versus P. We can see that the acceleration rate decreases with P, which is expected. Compared to the exact method, MECALS accelerates by 7$\sim$61$\times$ for different P's on different circuits. The experiment also shows the effectiveness of combining exact and approximate PBDs. For some $P < 100$ (*e.g.*, $P = 40$), they can achieve almost the same accuracy as $P = 100$, which corresponds to only using exact PBDs, while being much faster.

B. Synthesis Quality

In this section, we first compare the proposed ALS flow based on MECALS with MUSCAT [13] (a SOTA method proposed recently) under the WCE constraint. Tested benchmarks are listed in Table I. For the benchmarks also tested by MUSCAT, the selected WCE bounds are exactly the same as those used in MUSCAT. Actually, there are tens of WCE bounds for each benchmark, and each bound is tested in this experiment. Since there are many configurations in MUSCAT that produce approximate circuits of different qualities, we select the best one from the MUSCAT paper so that the largest area saving is achieved. For the benchmarks not tested by MUSCAT [13], we choose four different WCE bounds according to the PO number O, *i.e.*, $2^{\lfloor O/8 \rfloor}, 2^{\lfloor O/4 \rfloor}, 2^{\lfloor 3O/8 \rfloor}$ and $2^{\lfloor O/2 \rfloor}$. Additionally, different P values are chosen according to the circuit size. From Fig. 4, recall is almost 100% when $P = 40$. Thus, we choose $P = 40$ for all the circuits except the large ones, *i.e.*, *binsqrd*, *add128*, *mult16*, and *mult32*. To reduce runtime of larger circuits, P is set as 4 for *binsqrd*, *add128*, and *mult16*, and 0.4 for *mult32*.

TABLE II. Comparison of MECALS-based flow with MUSCAT. N/A: unable to obtain approximate circuits in 24 hours. A **bold** entry means that MECALS-based flow outperforms MUSCAT.

Circuit	Mean area saving		Mean delay saving		Runtime/s	
	MUSCAT	MECALS	MUSCAT	MECALS	MUSCAT	MECALS
absdiff	47.0%	**67.6%**	26.6%	**42.5%**	67	**3**
add8	55.9%	**58.9%**	45.2%	**53.3%**	15	**1**
add32	40.3%	**47.7%**	45.8%	28.4%	29	611
binsqrd	9.8%	**27.0%**	5.5%	**8.2%**	31546	**791**
buttfly	15.1%	**24.0%**	4.4%	**17.7%**	8	51
mac	10.0%	**32.2%**	5.7%	**13.9%**	3	8
mult8	61.5%	**75.8%**	38.7%	**53.3%**	7380	**1548**
Mean of the above	34.2%	**47.6%**	24.6%	**31.0%**	5578	**431**
add128	N/A	26.1%	N/A	4.2%	>24h	**17581**
mult16	N/A	14.7%	N/A	10.8%	>24h	**4526**
mult32	N/A	11.9%	N/A	2.7%	>24h	**44512**

Area (*resp. Delay*) *saving*, defined as the reduced area (*resp.* delay) of the approximate circuit over the area (*resp.* delay) of the exact circuit, is used to evaluate the approximate designs. Apparently, we prefer larger area and delay savings. We obtain average area saving, average delay saving, and average runtime over all the tested WCE bounds for each circuit. As shown in Table II, MUSCAT cannot generate approximate designs for *add128*, *mult16*, and *mult32* within 24 hours, while our method can deal with all of them. Compared with MUSCAT, on average, our method saves more area and delay with a relative improvement of 39.2% and 26.0%, respectively, and accelerates by 13×. Notably, our method always reduces more area than MUSCAT, and reduces more delay on all benchmarks except *add32*. This improvement is due to the ability of MECALS to handle complex LACs like SASIMI LAC, which are not supported in MUSCAT.

Moreover, we also approximate several adders and multipliers under the MaxSE constraint to show the wide applicability of our method. Since MUSCAT does not test the MaxSE constraint, our flow is compared with the truncating-based method, which sets some least significant bits of an exact circuit to 0. For each benchmark, we generate 5 different truncating circuits and measure their MaxSE normalized to the maximum possible square error $(2^O - 1)^2$, where O is the PO number. The number of truncation bits and the normalized MaxSEs are shown in Fig. 5. Then, we apply MECALS-based flow to generate approximate designs using the MaxSEs of the truncating circuits as the error bounds. As shown in Fig. 5, our flow always saves more area and delay than the truncating-based method. In most cases, both area and delay savings of a circuit produced by our flow increase with the MaxSE. Notably, for *mult16*, our flow reduces area and delay by 47% and 31% under a normalized MaxSE of no more than 6×10^{-8}.

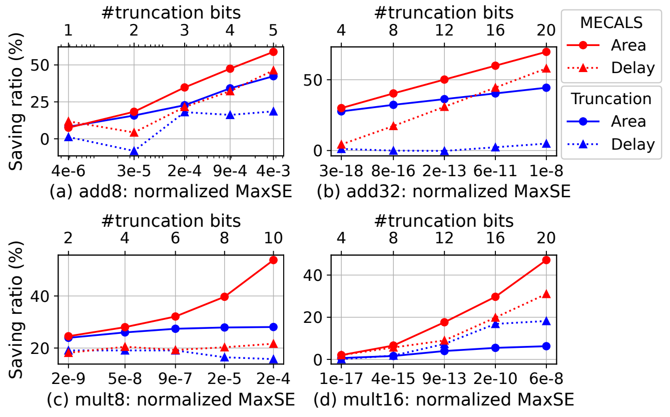


Fig. 5. Comparison between MECALS-based flow and truncating-based method under the MaxSE constraint.

VI. Conclusion

This paper proposes MECALS, a maximum error checking technique for approximate logic synthesis. It relies on an error model with PBDs and fast error checking with SAT sweeping. We also design an efficient ALS flow based on MECALS. The experimental results show that our flow improves the quality of ALS significantly and accelerates ALS dramatically.

References

[1] J. Han and M. Orshansky, “Approximate computing: An emerging paradigm for energy-efficient design,” in *ETS*, 2013, pp. 1–6.
[2] I. Scarabottolo *et al.*, “Approximate logic synthesis: A survey,” *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2195–2213, 2020.
[3] D. Shin and S. K. Gupta, “A new circuit simplification method for error tolerant applications,” in *DATE*, 2011, pp. 1–6.
[4] S. Venkataramani *et al.*, “Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits,” in *DATE*, 2013, pp. 1367–1372.
[5] Y. Wu *et al.*, “An efficient method for multi-level approximate logic synthesis under error rate constraint,” in *DAC*, 2016, pp. 1–6.
[6] S. Hashemi *et al.*, “BLASYS: Approximate logic synthesis using boolean matrix factorization,” in *DAC*, 2018, pp. 1–6.
[7] C. Meng *et al.*, “ALSRAC: Approximate logic synthesis by resubstitution with approximate care set,” in *DAC*, 2020, pp. 1–6.
[8] A. Chandrasekharan *et al.*, “Approximation-aware rewriting of AIGs for error tolerant applications,” in *ICCAD*, 2016, pp. 1–8.
[9] J. Schlachter *et al.*, “Design and applications of approximate circuits by gate-level pruning,” *IEEE Transactions on Very Large Scale Integration Systems*, vol. 25, no. 5, pp. 1694–1702, 2017.
[10] M. Češka *et al.*, “Approximating complex arithmetic circuits with formal error guarantees: 32-bit multipliers accomplished,” in *ICCAD*, 2017, pp. 416–423.
[11] I. Scarabottolo *et al.*, “Circuit carving: A methodology for the design of approximate hardware,” in *DATE*, 2018, pp. 545–550.
[12] ——, “A formal framework for maximum error estimation in approximate logic synthesis,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 4, pp. 840–853, 2022.
[13] L. Witschen *et al.*, “MUSCAT: MUS-based circuit approximation technique,” in *DATE*, 2022, pp. 1–6.
[14] R. Venkatesan *et al.*, “MACACO: Modeling and analysis of circuits for approximate computing,” in *ICCAD*, 2011, pp. 667–673.
[15] A. C. Chiang *et al.*, “Path sensitization, partial Boolean difference, and automated fault diagnosis,” *IEEE Transactions on Computers*, vol. 100, no. 2, pp. 189–195, 1972.
[16] Q. Zhu *et al.*, “SAT sweeping with local observability don’t-cares,” in *Advanced Techniques in Logic Synthesis, Optimizations and Applications*, Springer, 2011, pp. 129–148.
[17] A. Mishchenko *et al.*, *ABC: A system for sequential synthesis and verification*, http://people.eecs.berkeley.edu/~alanmi/abc/, 2022.
[18] Nangate, Inc., *Nangate 45nm open cell library*, https://si2.org/open-cell-library/, 2022.
[19] Brown University Scale Lab, *BACS: Benchmarks for approximate circuit synthesis*, https://github.com/scale-lab/BACS, 2022.