HEDALS: Highly Efficient Delay-driven Approximate Logic Synthesis

Chang Meng, Zhuangzhuang Zhou, Yue Yao, Shuyang Huang, Yuhang Chen, Weikang Qian, *Senior Member, IEEE*

***Abstract*—Approximate computing is an emerging paradigm for error-tolerant applications. By introducing a reasonable amount of inaccuracy, both the area and delay of a circuit can be reduced significantly. To produce approximate circuits automatically, many approximate logic synthesis (ALS) algorithms are proposed. However, they mainly focus on area reduction and are not optimal in reducing the circuit delay. In this paper, we propose HEDALS, a highly efficient delay-driven ALS framework, which supports various types of local approximate changes (LACs), circuit representations, and average error metrics. To reduce delay, HEDALS builds a critical error graph (CEG) consisting of nodes on the critical paths and error information, and finds an optimized set of LACs in the CEG by either a maximum flow-based method or a priority cut-based method. The resulting set of LACs is applied to shorten all critical paths simultaneously so that the circuit delay is reduced. Besides, the simultaneous application of multiple LACs also makes HEDALS extremely fast. Compared to a state-of-the-art method, on average, HEDALS can reduce the circuit delay by 32.3%, while being $167\times$ faster. The code of HEDALS is made open-source.**

***Index Terms*—approximate logic synthesis, approximate computing, delay optimization, maximum flow, priority cut**

I. Introduction

As the transistor size shrinks into nanoscale, it has been increasingly difficult to improve the performance and energy consumption of circuits by conventional design methods [1]. In addition, many recent applications are error-tolerant by their nature, such as machine learning, image processing, and multimedia. Under this circumstance, *approximate computing* was proposed as a novel circuit design paradigm [2]. Its basic idea is to modify the function of a target circuit without affecting its functionality at the application level. With carefully designed modifications, the resulting circuit will have a smaller area, delay, and power than the original version. To generate approximate circuits automatically, *approximate logic synthesis* (*ALS*) approaches are proposed, seeking to synthesize an optimized approximate circuit that satisfies a given error constraint.

Significant progress has been made in ALS in recent years [3]–[11], most of which mainly focused on reducing the circuit area. Although delay is usually also decreased as a byproduct of these ALS methods, the potential of ALS in optimizing delay has not been fully explored. Meanwhile, many error-tolerant applications have stringent timing constraints, such as real-time signal processing. For them, delay, instead of area, is the primary concern. Therefore, a delay-oriented ALS flow is preferred for these applications.

Given the difficulty in optimizing approximate circuits globally, existing ALS methods usually repeatedly apply *local approximate changes* (*LACs*) to the nodes in a circuit, such as replacing nodes by constant 0s or 1s, until the given error constraint is reached. In these methods, where area is the primary optimization target, all nodes in a circuit are treated as candidates for approximation, since they contribute to the area equally regardless of their locations. However, it is not the case for delay optimization. Circuit delay is determined by the critical paths. Thus, instead of considering all nodes in a circuit, effort should be put into the nodes on the critical paths. Furthermore, even approximating some nodes on the critical paths may be ineffective, as there may exist multiple critical paths with the same delay. If the approximation only reduces the delays of some, but not all, critical paths, the overall circuit delay remains unchanged. Instead, a better choice to reduce the circuit delay is to shorten all the critical paths simultaneously.

To address the above issues, we propose *HEDALS*, a highly efficient delay-driven approximate logic synthesis framework. To effectively reduce the delay, it only focuses on the nodes on the critical paths and only considers the LACs applied on these nodes. Moreover, it selects an optimized set of LACs and applies them simultaneously to the circuit so that all the critical paths are shortened, while the induced error is minimized.

Our main contributions are as follows:

1) To facilitate the finding of an optimized set of LACs that can reduce the circuit delay, while minimizing the induced error, we propose a novel data structure called *critical error graph* (*CEG*).
2) We propose a maximum flow-based method to find an optimized LAC set with the help of CEG and an efficient model to estimate the error caused by a set of LACs.
3) We propose a priority cut-based method to further improve the LAC set found by the maximum flow-

Chang Meng and Yuhang Chen are with the University of Michigan-Shanghai Jiao Tong University Joint Institute (UM-SJTU JI), Shanghai Jiao Tong University, China (emails: {changmeng, yuhang.chen}@sjtu.edu.cn).

Zhuangzhuang Zhou is with the Computer Systems Laboratory, Cornell University, USA (email: zz586@cornell.edu).

Yue Yao is with the School of Computer Science, Carnegie Mellon University, USA (email: yueyao@cs.cmu.edu).

Shuyang Huang is with the Amazon Inc., USA (email: shuyah@amazon.com).

Weikang Qian is with the UM-SJTU JI and the MoE Key Laboratory of Artificial Intelligence, Shanghai Jiao Tong University, China. He is also affiliated with the State Key Laboratory of ASIC & System, Fudan University, China (email: qianwk@sjtu.edu.cn).

Corresponding author: Weikang Qian.

This work is supported by the National Key R&D Program of China under grant number 2021ZD0114701 and the State Key Laboratory of ASIC & System Open Research Grant 2019KF004.

based method.
4) Based on the above techniques, we propose HEDALS, a delay-oriented ALS framework supporting various LAC types, circuit representations, and average error metrics.

The experimental results show that HEDALS can rapidly synthesize approximate circuits with significantly reduced delays. Besides, it has good scalability and wide applicability in terms of the supported LAC types, circuit representations, and error metrics. Compared to a state-of-the-art method, on average, HEDALS can reduce the circuit delay by 32.3%, while being 167× faster. The code of HEDALS is made open-source at https://github.com/SJTU-ECTL/HEDALS.

A preliminary version of this work was published in [12]. Compared to it, this work improves the error estimation model used in the maximum flow-based method. Furthermore, we propose a more effective priority cut-based method to find an optimized LAC set, leading to an improved circuit quality. Also, we conduct extensive experimental studies to demonstrate the scalability and wide applicability of HEDALS.

The rest of the paper is organized as follows. Section II discusses the related works. Section III introduces the background. Sections IV–VI first overview the HEDALS framework and then present its details. Section VII shows the experimental results. Finally, Section VIII concludes the paper.

II. Related Works

This section reviews some recent advances in ALS. A more comprehensive survey can be found in [13].

Most existing ALS works focus on area optimization [3]–[9]. They simplify the circuit by applying LACs, which are local modifications on sub-circuits. Shin and Gupta proposed a LAC that replaces a node in the circuit by a constant 0 or 1, which can be further propagated to simplify both the transitive fanin and fanout cones of the node for further area reduction [3]. Venkataramani *et al.* proposed a LAC that substitutes a node u in the circuit with another node v or its negation [4]. Then, the *maximum fanout-free cone* (*MFFC*) [14] of node u can be removed to reduce area. Wu and Qian proposed an approximate node simplification technique [5]. Its LAC is deleting some literals from the Boolean expression of a node in the circuit, which can lead to area reduction after technology mapping. Liu and Zhang proposed a stochastic ALS framework including some simple LACs like removing a gate and adding a gate [6]. Meng *et al.* proposed an ALS flow called *ALSRAC* [7]. Its LAC is approximately resubstituting a target node in the circuit by a new function on some existing nodes in the circuit. Then, removing the MFFC of the target node leads to area reduction. Witschen *et al.* formulated the ALS problem as a minimal unsatisfiable subset problem for solving [8]. Its LAC is also replacing a node in the circuit by a constant 0 or 1 [3]. Ma *et al.* proposed an ALS method called *BLASYS* [9]. Its LAC is approximately decomposing a sub-circuit into two smaller units, *i.e.*, a compressor and a decompressor, by Boolean matrix factorization. We note that HEDALS can handle all the above LACs except the one proposed in [9]. Their common feature is that they modify a sub-circuit with a single output. However, the LAC proposed in [9] affects a sub-circuit that may have multiple outputs, and it is not supported by HEDALS.

There are also few works that can directly reduce the delay of approximate circuits [10], [11]. In [10], Venkataramani *et al.* proposed an ALS method called *SALSA*. It identifies the don't cares based on the error constraint and then converts the ALS problem into a traditional logic synthesis problem with don't cares. If a traditional delay-oriented logic synthesis method is applied, then the delay of the approximate circuit will be reduced. However, SALSA is not applicable to large circuits with many *primary inputs* (*PIs*), since the don't cares are expressed in terms of PIs, and a large circuit may have an enormous number of such don't cares. In [11], Chandrasekharan *et al.* proposed an ALS method based on rewriting of *AND-inverter graph* (*AIG*). To reduce the delay, for each critical path in an AIG, a cut of a node is selected and rewritten with a constant 0. Although this change shortens the critical path where the node locates, the delay of the other critical paths may keep the same. Besides, the selection of the cut is solely based on the cut size without considering the error induced by the change. Thus, it may select a sub-optimal LAC.

Compared to the above works that can directly reduce the circuit delay, HEDALS is a scalable framework that can handle large circuits. In addition, it selects a set of cuts on the critical paths for approximation to ensure delay reduction. Furthermore, it applies a more sophisticated strategy for cut selection by taking the introduced errors into account.

III. Background

In this section, we introduce the background of HEDALS.

A. Circuit Terminologies

We focus on multi-level combinational circuits, which can be modeled as a directed acyclic graph. In a circuit, a PI is a node without any fanin. A *functional node* is a node performing logic operations. A PO is a dummy node driven by a functional node or a PI. It has a single fanin and no fanout. A *path* is a sequence of connected nodes in the circuit.

A circuit can be represented in various forms. In an AIG, functional nodes are two-input AND gates, and edges can be either complemented or non-complemented, where a complemented edge indicates the negation of the signal. For example, Fig. 1 shows an AIG with 5 PIs $x_0, x_1, \ldots, x_4$, 6 functional nodes (two-input AND gates) $n_0, n_1, \ldots, n_5$, and 2 POs y_0 and y_1. In a *majority-inverter graph* (*MIG*), functional nodes are three-input majority nodes, and edges can also be either complemented or non-complemented. In a *lookup table* (*LUT*) network, which is the underlying representation for an FPGA design, functional nodes are LUTs. In a gate netlist, functional nodes are gates. For both LUT network and gate netlist, edges can only be non-complemented. HEDALS supports the above commonly-used circuit representations.

B. Circuit Timing

In this work, when calculating delays, we only consider node delays and ignore wire delays. The method

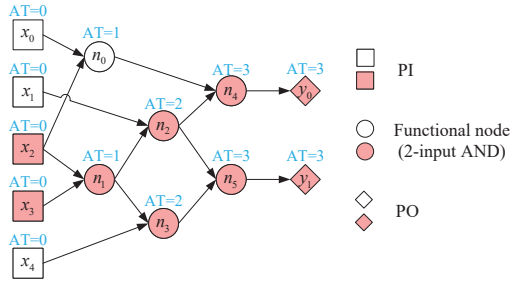


Fig. 1. An example circuit in the AIG representation, where all the edges are non-complemented. The ATs of the nodes are marked in blue. The red nodes are on the critical paths.

of calculating the node delay varies for different circuit representations. In AIGs, MIGs, and LUT networks, the node delay is set as 1. In gate netlists, where each node is a gate from a standard cell library, the delay of a node can be looked up from the library according to its input transition time and output capacitance. The *arrival time* (*AT*) of a node is the largest delay of all paths from a PI to the node. A path from a PI to a PO with the maximum delay is a *critical path*, and the maximum delay is the *circuit delay*. Note that a circuit can have multiple critical paths. For example, for the AIG in Fig. 1, the AT of each node is shown in blue, and the red nodes are on the critical paths. This AIG has more than one critical path. The ATs, circuit delay, and critical paths can be obtained by *static timing analysis* (*STA*) [15].

C. Error Metrics

Among the various error metrics for evaluating the accuracy of approximate circuits, HEDALS supports commonly-used average error metrics. Let $\boldsymbol{y} : \mathbb{B}^I \to \mathbb{B}^O$ and $\hat{\boldsymbol{y}} : \mathbb{B}^I \to \mathbb{B}^O$ be the multiple-output Boolean functions of an exact circuit and an approximate circuit, respectively. Assume that the numbers of PIs and POs of the circuits are I and O, respectively. The average error is defined as the average deviation between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ over all input patterns:

$$\text{average\_error} = \sum_{\boldsymbol{x} \in \mathbb{B}^I} D(\boldsymbol{y}(\boldsymbol{x}), \hat{\boldsymbol{y}}(\boldsymbol{x})) \cdot p(\boldsymbol{x}),$$

where $\boldsymbol{y}(\boldsymbol{x})$ and $\hat{\boldsymbol{y}}(\boldsymbol{x})$ are binary vectors of length O, denoting the outputs of the exact and the approximate circuits under the input pattern $\boldsymbol{x}$, respectively, $p(\boldsymbol{x})$ is the occurring probability of the pattern $\boldsymbol{x}$, and D is a distance function measuring the deviation between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$.

Typical average errors include *error rate* (*ER*), *mean error distance* (*MED*), and *mean Hamming distance* (*MHD*). ER is the probability of an input pattern producing a wrong output for the approximate circuit. Its distance function $D_{\text{ER}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = 1$ if $\boldsymbol{y} \neq \hat{\boldsymbol{y}}$ and 0 otherwise.

MED is the mean absolute difference between the numerical values encoded by the outputs of exact and approximate circuits. Its distance function is

$$D_{\text{MED}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = |int(\boldsymbol{y}) - int(\hat{\boldsymbol{y}})|,$$

where the function $int(\boldsymbol{v})$ returns the integer encoded by the binary vector $\boldsymbol{v}$. MHD is the average number of bit-flips in $\hat{\boldsymbol{y}}$ with respect to the original $\boldsymbol{y}$. Its distance function is

$$D_{\text{MHD}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{i=0}^{O-1} |y_i - \hat{y}_i|,$$

where y_i and $\hat{y}_i \in \{0, 1\}$ denote the i-th PO of the exact and the approximate circuits, respectively.

Average errors are usually estimated with Monte Carlo simulation by sampling a set of M input patterns $\mathbb{X} = \{\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^M\}$. That is,

$$\text{average\_error} \approx \frac{1}{M} \sum_{\boldsymbol{x} \in \mathbb{X}} D(\boldsymbol{y}(\boldsymbol{x}), \hat{\boldsymbol{y}}(\boldsymbol{x})).$$

In addition, the *normalized* MED (NMED) and *normalized* MHD (NMHD) are defined as follows:

$$\text{NMED} = \frac{\text{MED}}{2^O - 1}, \text{NMHD} = \frac{\text{MHD}}{O}.$$

IV. Overview of HEDALS

Given an exact circuit, HEDALS aims at synthesizing an approximate circuit with reduced delay without increasing its area. Notably, the circuit can be represented in a commonly-used form such as AIG, MIG, LUT network, and gate netlist. The error constraint can be any average error constraint, such as ER, MED, and MHD.

Algorithm 1: The procedure of HEDALS.

Input: an exact circuit G and an error upper bound e_b.
Output: an approximate circuit G_{apx} with an error $\leq e_b$.
1 $G_{apx} \Leftarrow G$; $HasSol \Leftarrow true$;
2 **while** $HasSol$ **do** ▷ can further reduce delay
3 $\quad (HasSol, G_{apx}) \Leftarrow FindApplyOptLACSet(G_{apx}, e_b)$;
4 $G_{apx} \Leftarrow SynthesizeAndMap(G_{apx})$;
5 **return** G_{apx};

The overall flow of HEDALS is shown in Algorithm 1. Its inputs are an exact circuit G and an error upper bound e_b. It outputs an approximate circuit G_{apx} with an error not exceeding e_b. Line 1 initializes the approximate circuit G_{apx} as the exact circuit G and a Boolean variable *HasSol* as true. Then, Lines 2–3 iteratively apply sets of LACs to simplify the approximate circuit G_{apx}. Finally, Line 4 applies traditional delay-oriented logic synthesis and technology mapping to produce a mapped approximate circuit G_{apx}, which is then returned (Line 5).

In each iteration, the function *FindApplyOptLACSet* is called to do the simplification (Line 3). Its main task is to find a set of LACs, marked as L, and apply them to the approximate circuit G_{apx} so that the circuit delay decreases by some amount. For a LAC set C, we use $Error(C)$ to denote the error of the approximate circuit obtained by applying the LACs in set C. We also refer to it as the *error of the LAC set* C. Since the LAC set L found in each iteration reduces the circuit delay by some amount, to minimize the delay of the final approximate circuit, a good heuristic is to maximize the number of iterations. To achieve this, we aim at minimizing the error of the LAC set L in each iteration. Thus, we formulate a problem as follows:

Problem 1 *Given the set $\mathbb{L}$ of all possible LAC sets in an approximate circuit G_{apx}, find the LAC set $L \in \mathbb{L}$ with the minimum error, while also satisfies that*

1) *after applying all LACs in L, the circuit delay is reduced;*
2) *$Error(L)$ is no more than the error bound e_b.*

Note that the solution space of Problem 1 is very large, since it includes all possible LAC sets in the approximate circuit G_{apx}. On the one hand, there are many sets of nodes in a circuit, on which LACs can be identified. On the other hand, even for a certain node set, there are many ways of

introducing LACs to the nodes in the set, since each node in the set generally has multiple LACs.

The function *FindApplyOptLACSet* provides an efficient solution to Problem 1. To reduce the solution space, it exploits a novel data structure, *critical error graph* (*CEG*), which will be elaborated in Section V. Using the CEG, we propose two implementations of *FindApplyOptLACSet* to solve Problem 1, *i.e.*, a maximum flow-based method and a priority cut-based method. Their details will be described in Section VI. Both methods first pre-process the current approximate circuit and then find an optimized LAC set from the pre-processed circuit. If an optimized set L is found, then the function *FindApplyOptLACSet* sets the variable *HasSol* to true, applies all LACs in L to simplify the pre-processed circuit, and returns the resulting circuit (Line 3). However, it is also possible that for a pre-processed circuit, no LAC set satisfying the conditions in Problem 1 can be found. Then, the function *FindApplyOptLACSet* sets *HasSol* to false and returns the pre-processed circuit. In this case, the loop is terminated.

V. Critical Error Graph

This section presents the details of CEG, which is a key data structure used in the function *FindApplyOptLACSet*. Since the solution to Problem 1 should be a LAC set that can reduce the circuit delay, we first analyze which LAC sets can achieve this in Section V-A. However, as the number of LAC sets that can reduce delay is large, we propose to focus on a promising subset of these LAC sets in Section V-B, which naturally leads to CEG. We finally conclude how CEG is built in Section V-C.

A. LAC Sets for Delay Reduction

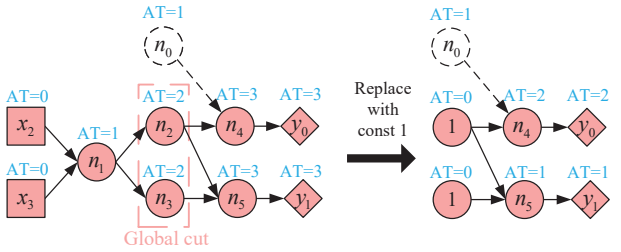


Fig. 2. An example of shortening all the critical paths in the critical graph by applying delay-reducing LACs to the nodes in a global cut of the critical graph. The left figure is the critical graph of the AIG in Fig. 1.

Clearly, in order to effectively reduce the delay, we only need to focus on the nodes on the critical paths. They form a *critical graph* formally defined below.

Definition 1 *The* **critical graph** $CG = (V; E)$ *for a circuit is a subgraph of it, where V and E are the sets of nodes and edges, respectively, on the critical paths of the circuit.*

If a node in the critical graph corresponds to a PI/PO/functional node in the original circuit, we also call it a PI/PO/functional node in the critical graph.

Example 1 *The left part of Fig. 2 shows the critical graph for the AIG in Fig. 1, where the set of nodes $V = \{x_2; x_3; n_1; n_2; n_3; n_4; n_5; y_0; y_1\}$ forms the critical paths of the AIG. In the critical graph, the PIs are x_2 and x_3, the POs are y_0 and y_1, and the functional nodes are $n_1; \ldots; n_5$. Note that the node n_0 does not belong to the critical graph; it is shown to illustrate the computation of AT clearly.*

Because there usually exist multiple critical paths in a circuit, thus, to reduce the circuit delay, it is not enough to shorten one critical path. Instead, we should shorten all the critical paths. A *global cut* gives us a way to achieve this.

Definition 2 *A* **global cut** *of the critical graph is a set of nodes satisfying: 1) each node in the set is a functional node,* i.e.*, neither a PI nor a PO, and 2) each path from a PI to a PO of the critical graph passes at least one node in the set.*

For instance, $\{n_2; n_3\}$ is a global cut of the critical graph shown in the left part of Fig. 2. If for each node n in a global cut, we apply a LAC on node n such that n's AT decreases, then all the critical paths are shortened, and the circuit delay is reduced. Such a LAC on a node n leading to a reduction of n's AT is called a *delay-reducing LAC* of node n.

Example 2 *Suppose that we apply the constant LAC [3] to the circuit shown in the left part of Fig. 2. By replacing each node in the global cut $\{n_2; n_3\}$ with a constant 1, the ATs of n_2 and n_3 are both reduced to 0. Even without further applying constant propagation, the AT of the PO y_0 is reduced from 3 to 2, and that of the PO y_1 is reduced from 3 to 1. Hence, the circuit delay is reduced to 2.*

The above example shows that constant LAC is delay-reducing. Besides it, most LACs proposed in previous works [4]–[8] (discussed in Section II) are also delay-reducing and hence, are applicable to HEDALS.

By the above analysis, we can conclude that a set of delay-reducing LACs applied to the nodes on a global cut of the critical graph of the circuit can reduce the circuit delay. We call such a LAC set a *delay-reducing LAC set*.

B. Promising Subset of All Delay-reducing LAC Sets and Critical Error Graph

The optimal solution to Problem 1 must be a delay-reducing LAC set. However, the number of delay-reducing LAC sets is usually very large. First, for a single global cut, its number of delay-reducing LAC sets grows exponentially with the size of the cut, as each node in the cut may have multiple delay-reducing LACs. Moreover, for a large circuit, the number of global cuts of its critical graph is usually large. Thus, the number of delay-reducing LAC sets is very large, and it is prohibitive to evaluate all of them.

To reduce the complexity, we try to identify a promising subset of all delay-reducing LAC sets with small errors.[1] To achieve this purpose, for each node n in the critical graph, after obtaining the errors of all of its delay-reducing LACs, we only keep the one with the minimum error. We call it the *min-error LAC* of node n. Then, for each global cut N of the critical graph, we only maintain a single delay-reducing LAC set that consists of the min-error LAC for each node on the cut. We call it the *leading LAC set* of cut N. The leading LAC sets of all the global cuts of the critical graph

[1]Note that the error metric can be any average error metric of interest, such as ER, MED, and MHD, which can be calculated as shown in Section III-C.

form the promising subset. By focusing on the leading LAC sets, we can convert Problem 1 into the following one.

Problem 2 *Given an approximate circuit G_{apx}, among all the leading LAC sets of all the global cuts of the critical graph of G_{apx}, find one with the minimum error, while also satisfying that the error does not exceed the error bound e_b.*

Since the leading LAC set of any global cut is a delay-reducing LAC set, the circuit delay is guaranteed to be reduced after applying all LACs in the final solution set. Thus, the delay constraint in Problem 1 does not appear in Problem 2.

An essential component of Problem 2 is the critical graph of the circuit with each node in the graph associated with its min-error LAC. This leads to the following definition of the critical error graph (CEG) of a circuit. Specifically, suppose that before applying any LAC, the error of the current approximate circuit, which we refer to as the *base error*, is e_{base}. After applying the min-error LAC of node n, the error of the resulting approximate circuit is e. The increased error caused by the LAC is $e = e \quad e_{base}$, which is also called the *minimum error increase (MEI)* of node n. The *CEG* of the approximate circuit is the critical graph of the circuit with each functional node associated with its MEI. For example, Fig. 3 shows the CEG of the AIG in Fig. 1, where the MEI of each functional node is put near it. CEG facilitates the solving of Problem 2, which will be described in Section VI.

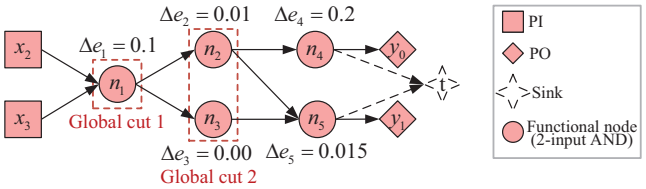


Fig. 3. The critical error graph (CEG) of the AIG in Fig. 1. The value Δe_i beside the functional node n_i represents the minimum error increase (MEI) of n_i, where the error metric can be any average error metric of interest, such as ER, MED, and MHD. Note that the sink node t does not belong to the CEG; it is only used for obtaining a set of global cuts in Section VI-B.

C. Building Critical Error Graph

To build the CEG of an approximate circuit G_{apx}, we first build the critical graph *CG* from the nodes and the edges on the critical paths of G_{apx}. Then, for each functional node in *CG*, a set U of delay-reducing LACs is obtained. After that, we obtain the error of applying each LAC $l \; 2 \; U$ to the circuit G_{apx}, which is achieved by an efficient error estimation method, VECBEE [16]. Then, we can obtain the min-error LAC and the corresponding MEI of each functional node in *CG* to finally build the CEG.

VI. Obtaining an Optimized LAC Set

This section presents the details of the function *FindApplyOptLACSet*, which provides a solution to Problem 2. For a large circuit with many global cuts of its critical graph, it is still very challenging to exactly solve Problem 2 to find a leading LAC set with the minimum error. Instead, we present two methods to obtain an optimized leading LAC set with a low error. The first is a maximum flow-based method. The second is a priority cut based-method, which builds upon and improves the maximum flow-based method.

A. Maximum Flow-based Method

A solution to Problem 2 usually needs to obtain the error of a LAC set, which is typically done by time-consuming logic simulation. To improve the efficiency, the maximum flow-based method relies on an efficient linear model to estimate the error of a LAC set. We first introduce the model and then present the method using the model.

1) Estimation of Error of a LAC Set: Consider the leading LAC set $L = fl_1; l_2; : : : ; l_m g$ of a global cut with m nodes $N = fn_1; n_2; : : : ; n_m g$, where l_i (1 i m) is the min-error LAC of node n_i. Assume that before the LACs are applied, the circuit has a base error, e_{base}, and that after applying the LAC l_i alone, the resulting approximate circuit has an error of e_i. By definition, the MEI of node n_i is $e_i = e_i \quad e_{base}$. In the preliminary version of this work [12], we estimate the error of the leading LAC set L by a linear model as follows:

$$Error(L) \quad e_1 + e_2 + \quad + e_m : \tag{1}$$

Since $e_i = e_i \quad e_{base}$, we also have

$$\begin{aligned} Error(L) \quad & (e_{base} + \quad e_1) + \quad + (e_{base} + \quad e_m) \\ & = m e_{base} + \quad e_1 + \quad e_2 + \quad + \quad e_m : \end{aligned} \tag{2}$$

Clearly, the model accumulates the base error m times, which is unreasonable. In this work, we improve it as follows:

$$Error(L) \quad e_{base} + \quad e_1 + \quad e_2 + \quad + \quad e_m : \tag{3}$$

The new model is more accurate, as the example below shows.

Example 3 *In the critical graph shown in Fig. 3, consider two global cuts $N_1 = fn_1 g$ and $N_2 = fn_2; n_3 g$. Assume that the LACs $l_1; l_2$, and l_3 are the min-error LACs of the nodes n_1, n_2, and n_3, respectively. Then, the leading LAC sets of N_1 and N_2 are $L_1 = fl_1 g$ and $L_2 = fl_2; l_3 g$, respectively. Assume that the base error e_{base} is 0.5, and that the MEIs of n_1, n_2, and n_3 are $e_1 = 0{:}1$, $e_2 = 0{:}01$, and $e_3 = 0$, respectively. Note that LAC l_3 is an exact local change introducing no error.*

By Eq. (2), the errors of L_1 and L_2 calculated by the previous model are $Error_{old}(L_1) = 0{:}5 + 0{:}1 = 0{:}6$ and $Error_{old}(L_2) = 2 \quad 0{:}5 + 0{:}01 + 0 = 1{:}01$, respectively, which indicates that the LAC set L_1 is better due to its smaller estimated error. However, since LAC l_3 is an exact local change, the error of $L_2 = fl_2; l_3 g$ equals that caused by applying l_2 only. Thus, the actual error of L_2 is $Error_{actual}(L_2) = 0{:}5 + 0{:}01 = 0{:}51$. Besides, the actual error of $L_1 = fl_1 g$ is $Error_{actual}(L_1) = 0{:}5 + 0{:}1 = 0{:}6$. Therefore, the LAC set L_2 is better in reality.

Now, applying our proposed model, i.e., Eq. (3), we can obtain the errors of L_1 and L_2 as $Error_{new}(L_1) = 0{:}5 + 0{:}1 = 0{:}6$ and $Error_{new}(L_2) = 0{:}5 + 0{:}01 + 0 = 0{:}51$, respectively. Thus, L_2 is better, which agrees with the real situation. Furthermore, for this example, the errors estimated by our proposed model equal the actual values.

It should be noted that the proposed estimation may not be accurate, since the exact error increase of applying multiple LACs together may not equal the sum of the error increases of applying each individual LAC alone. Nevertheless, for the average error metrics such as ER, MED, and MHD, the sum is still a good first-order approximation and enables the design of a more efficient algorithm.

2) Entire Flow of Maximum Flow-based Method: For simplicity, we call the sum of the MEIs of all nodes in a global cut *the MEI sum* of the global cut. By our error estimation model shown in Eq. (3), the error of the leading LAC set of a global cut equals the MEI sum of the global cut plus the base error. Since the base error is the same for all the leading LAC sets, Problem 2 is converted into selecting the optimal global cut from the CEG with the minimum MEI sum.

To find the optimal global cut, our method first maps the original CEG into a dual graph and then solves a network flow problem on it. The dual graph is built as follows.

(a) For each functional node n with MEI e in the CEG, we add a pair of nodes n_a and n_b and an edge from n_a to n_b with a capacity of e to the dual graph.
(b) For each edge from a functional node u to a functional node v in the CEG, we add an edge from u_b to v_a with an infinite capacity to the dual graph.
(c) We add a source node s. For each edge from a PI node to a functional node n in the CEG, we add an edge from s to n_a with an infinite capacity to the dual graph.
(d) We add a sink node t. For each PO node in the CEG, let its only fanin functional node be n. We add an edge from n_b to t with an infinite capacity to the dual graph.

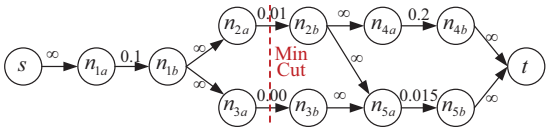


Fig. 4. The dual graph built from the CEG shown in Fig. 3.

The dual graph built from the CEG shown in Fig. 3 is given in Fig. 4. The dual graph is a classic flow network [17]. For a flow network, a *cut* is defined as a set of edges that disconnects the source and sink upon removal. The *capacity* of a cut is the total capacity of all edges in the cut. A *minimum cut* of a flow network is a cut with the minimum capacity over all cuts of the flow network. Given the above mapping procedure, it is easily seen that the problem of selecting the global cut of the CEG with the minimum MEI sum now reduces to the problem of finding a minimum cut in the dual graph. By the max-flow min-cut theorem [17], the capacity of a minimum cut in a flow network equals the maximum flow of the network. Thus, we can find a minimum cut of the dual graph by solving the maximum flow problem on the graph. Once each edge in the minimum cut is obtained, we can get the corresponding node in the CEG from the mapping relation and obtain the global cut in the CEG with the minimum MEI sum. By collecting the min-error LAC of each node in the global cut, we can obtain an optimized set of LACs.

A special case is that there may exist an edge with a negative capacity in the dual graph, which corresponds to a negative MEI of a node n in the CEG. This implies that the circuit error decreases after applying the min-error LAC of node n. Since a maximum flow algorithm cannot work on a flow network with negative capacities, a pre-processing is required, which corresponds to the pre-processing mentioned in Section IV. Note that the AT of node n is reduced by applying the min-error LAC of n. Such a LAC reducing both the error and the AT is highly desired. Thus, if there exists a node with a negative MEI in the CEG, we can directly apply the min-error LAC of the node to simplify the current approximate circuit and update the CEG. This process repeats until there is no negative MEI in the CEG. After that, a maximum flow algorithm can be applied.

Algorithm 2: *FindApplyOptLACSet_Flow*, a function for finding and applying an optimized LAC set to reduce delay by the maximum flow-based method.

Input: the current approximate circuit G_{apx} and an error upper bound e_b.
Output: a flag *HasSol*, denoting the existence of a valid LAC set, and a new approximate circuit G_{new}.

1 $G_{apx} \Leftarrow PreprocessNegativeCapacity(G_{apx})$;
2 $CEG \Leftarrow BuildCriticalErrorGraph(G_{apx})$;
3 $DualGraph \Leftarrow BuildDualGraph(CEG)$;
4 $MinCut \Leftarrow SolveMaxFlow(DualGraph)$;
5 Global cut $\mathcal{N} \Leftarrow EdgeToNode(MinCut)$;
6 LAC set $\mathcal{L} \Leftarrow GetMinErrorLACs(\mathcal{N})$;
7 **if** $GetError(\mathcal{L}) \leq e_b$ **then** ▷ if solutions exist
8 $\quad G_{new} \Leftarrow ApplyLACSet(G_{apx}, \mathcal{L})$; **return** $(true, G_{new})$;
9 **else** $G_{new} \Leftarrow G_{apx}$; **return** $(false, G_{new})$;

Algorithm 2 shows the entire flow of the maximum flow-based implementation of the function *FindApplyOptLACSet*, which finds an optimized set of LACs and applies them to simplify the approximate circuit. Line 1 pre-processes the current approximate circuit G_{apx} to avoid the occurrence of negative MEIs in the CEG. It repeatedly finds the node in the CEG with the smallest MEI and applies its min-error LAC to update the approximate circuit, until the MEIs of all nodes in the CEG of the approximate circuit are non-negative. Then, Line 2 constructs the CEG of the current approximate circuit G_{apx}. Line 3 builds the dual graph from the CEG. Line 4 solves the maximum flow problem and returns the minimum cut in the dual graph. Line 5 maps the minimum cut in the dual graph into the global cut N in the CEG. Line 6 obtains the set of min-error LACs L for all nodes in N. Then, we accurately evaluate the error of the LAC set L by logic simulation. If the error of L does not exceed the upper bound e_b, then the LACs in L are applied to the approximate circuit G_{apx} pre-processed in Line 1, and Line 8 returns *HasSol* = *true* and the resulting approximate circuit G_{new}. Otherwise, Line 9 returns *HasSol* = *false* and the pre-processed circuit G_{apx}.

B. Priority Cut-based Method

As we mentioned at the end of Section VI-A1, the error estimation model used in the maximum flow-based method is not accurate. This means that the LAC set found by the maximum flow-based method can be further improved. To achieve this, we propose a priority cut-based method in this section. For a set of nodes S in the CEG, we use $Error(S)$ to denote the error of the approximate circuit obtained by applying the min-error LACs of the nodes in set S. We also refer to it as the *error of the node set* S. Assume that the global cut found by the maximum flow-based method is N_{ref}. Let $= Error(N_{ref})$. We want to find a better global cut N so that $Error(N) <$. We first present a basic method to do this by traversing a set of global cuts, and then a pruning method for acceleration, which leads to the priority cut-based method.

1) Basic Method by Traversing a Set of Global Cuts: This method traverses a set of global cuts of the CEG to find whether there exists one with a lower error than N_{ref}. We next present how the set of global cuts is constructed. We first modify the CEG by adding a sink node t and connecting the fanin of each PO to node t. As shown in Fig. 3, the fanins of the POs are n_4 and n_5, and they are connected to the sink node t. The following definition of a local cut helps the construction.

Definition 3 *A* **local cut** *of a node n in the CEG is either a trivial set $\{n\}$, or a set of nodes satisfying: 1) each node in the set is a functional node, and 2) each path from a PI of the CEG to a fanin of node n passes at least one node in the set.*

With the sink node t added, by Definition 3, a local cut of node t except $\{t\}$ corresponds to a global cut of the CEG. Thus, we only need to obtain a set of local cuts of the sink node t. To achieve this, we modify the method proposed in [18] to derive a set of local cuts of each node n in the CEG, denoted as $\Phi(n)$. Our method relies on an operator $\bowtie$ for merging two sets of local cuts A and B:

$$A \bowtie B = \{a \cup b | a \in A, b \in B\}.$$

Particularly, $A \bowtie B$ is empty if either A or B is empty. In addition, if there are repetitive local cuts after performing $A \bowtie B$, only one of them is kept, and the others are discarded.

Example 4 *Consider two sets of local cuts $A = \{\{n_1\}, \{n_2\}\}$ and $B = \{\{n_1\}, \{n_3\}\}$. $A \bowtie B = \{\{n_1\}, \{n_1, n_2\}, \{n_1, n_3\}, \{n_2, n_3\}\}$.*

For each functional/sink node n in the CEG, assume that it has r fanins $u_1, u_2, \cdots, u_r$. The set of local cuts of node n, $\Phi(n)$, is obtained recursively as follows:

$$\Phi(n) = \begin{cases} \{\{n\}\}, \text{if } n \text{ is connected to any PI;} \\ \{\{n\}\} \cup (\Phi(u_1) \bowtie \cdots \bowtie \Phi(u_r)), \text{otherwise.} \end{cases} \quad (4)$$

A brief explanation of Eq. (4) is as follows. For a node n connected to any PI in a CEG, assume that one of the connected PIs is x. Then, the path from x to a fanin of n has only one node, which is x, a non-functional node. Thus, there does not exist a set of nodes satisfying both Conditions (a) and (b) in Definition 3, and hence, only the trivial set $\{n\}$ can be a local cut of n. For a node n not connected to any PIs in the CEG, all its fanins are functional nodes. In this case, a local cut of n can be either the trivial set $\{n\}$, or a local cut obtained by merging the local cuts of the fanins of n.

We apply Eq. (4) to each functional/sink node in the CEG in a topological order to finally get $\Phi(t)$. After that, a set of global cuts is obtained as $\Phi(t) \cup \{t\}$. An example of how to obtain a set of global cuts is as follows.

Example 5 *In Fig. 3, a set of local cuts of each functional/sink node can be obtained by Eq.* (4) *in a topological order as follows:*

$\Phi(n_1) = \{\{n_1\}\}$;
$\Phi(n_2) = \{\{n_2\}\} \cup \Phi(n_1) = \{\{n_1\}, \{n_2\}\}$;
$\Phi(n_3) = \{\{n_3\}\} \cup \Phi(n_1) = \{\{n_1\}, \{n_3\}\}$;
$\Phi(n_4) = \{\{n_4\}\} \cup \Phi(n_2) = \{\{n_1\}, \{n_2\}, \{n_4\}\}$;
$\Phi(n_5) = \{\{n_5\}\} \cup (\Phi(n_2) \bowtie \Phi(n_3))$
$= \{\{n_1\}, \{n_1, n_2\}, \{n_1, n_3\}, \{n_2, n_3\}, \{n_5\}\}$;
$\Phi(t) = \{\{t\}\} \cup (\Phi(n_4) \bowtie \Phi(n_5)) =$
$\{\{t\}, \{n_1\}, \{n_1, n_2\}, \{n_1, n_3\}, \{n_1, n_4\}, \{n_1, n_5\},$
$\{n_2, n_3\}, \{n_2, n_5\}, \{n_4, n_5\}, \{n_1, n_2, n_3\},$
$\{n_1, n_2, n_4\}, \{n_1, n_3, n_4\}, \{n_2, n_3, n_4\}\}$.

Then, a set of global cuts of the CEG is obtained as $\Phi(t) \cup \{t\}$.

For each obtained global cut, we need to run logic simulation to obtain its error and then pick the cut with the smallest error. Unfortunately, the basic method may consider a large number of global cuts. Assume that the set of functional nodes in the CEG is V. By Definition 2, a global cut is a subset of V. In the worst case, the number of global cuts considered by the basic method, N_b, can approach the total number of subsets of V. Thus, we have $N_b = O(2^{|V|})$, which makes the basic method impractical for a large circuit.

2) Accelerated Method: To make the basic method practical, we propose an accelerated method based on pruning.

Consider that in a CEG where all MEIs are non-negative, we merge two local cuts a and b, and the resulting local cut is $c = a \cup b$. We make an assumption that $Error(c)$ is larger than $Error(a)$ and $Error(b)$. This assumption is reasonable, since c is a superset of both a and b. Thus, compared to approximating a or b with the min-error LACs, approximating c with the min-error LACs modifies more nodes in the circuit. In addition, since all MEIs are non-negative, each min-error LAC applied will increase the error. Hence, approximating c is more likely to introduce a larger error. Under this assumption, when updating the set of local cuts of node n by Eq. (4), we can only keep the local cuts with errors no more than ϵ, *i.e.*, the minimum error obtained by the maximum flow-based method. The reason is that under the assumption, a global cut with error no more than ϵ cannot be the merge of some local cuts that include one cut with error greater than ϵ.

In practice, we introduce a new operator $\boxtimes_\epsilon$ that merges two sets of local cuts A and B with an error limit ϵ:

$$A \boxtimes_\epsilon B = \{a \cup b | a \in A, b \in B, Error(a \cup b) \leq \epsilon\}.$$

For a node set S, logic simulation is used to obtain $Error(S)$ accurately. In this way, only those global cuts with errors no more than ϵ are explored, and hence, we may find a better global cut with a smaller error than that obtained by the maximum flow-based method.

We call a local cut with its error no more than ϵ a *priority cut*. With the new operator $\boxtimes_\epsilon$, for each functional/sink node n in the CEG, we can rewrite Eq. (4) to the following one to obtain a set of priority cuts of n, denoted as $\Psi(n)$:

$$\Psi(n) = \begin{cases} \{\{n\}\}, \text{if } n \text{ is connected to any PI;} \\ \{\{n\}\} \cup (\Psi(u_1) \boxtimes_\epsilon \cdots \boxtimes_\epsilon \Psi(u_r)), \text{otherwise.} \end{cases} \quad (5)$$

We also apply Eq. (5) to each functional/sink node in the CEG in a topological order. After that, we obtain a set of global cuts with errors no more than ϵ as $\Psi(t) \cup \{t\}$, where t is the sink node of the CEG.

In addition, for efficiency concerns, it is undesired to have too many priority cuts in $\Psi(n)$ for each n. Thus, if there are more than P priority cuts in $\Psi(n)$, where P is a user-specified limit, we sort the priority cuts in $\Psi(n)$ in

the ascending order of their errors and only keep the top priority cuts. By considering more priority cuts with a larger , it is likely to find a better global cut with a smaller error, which leads to a better approximate design. Meanwhile, a larger also causes a longer runtime. Thus, there is a quality-runtime tradeoff by applying different 's. To decide a good choice of , we can select several representative benchmarks, test their quality-runtime tradeoff with various 's, and then choose a leading to circuits with good qualities in a short time.

Now, we analyze the number of cuts whose errors should be evaluated by logic simulation. Assume that the maximum fanin count for all functional/sink nodes in the CEG is . For each functional/sink node n in the CEG, Eq. (5) merges the priority cuts of n's fanins. Since there are at most priority cuts for each fanin of n, after merging these priority cuts, $O(\quad)$ new local cuts are generated on n. We need to evaluate the errors of these new local cuts and keep the best ones. Usually, is a small constant. For example, 4 for all circuits in the BACS [13] and ISCAS [19] benchmark suites. Thus, for each node, the accelerated method needs to run logic simulation for $O(\quad) = O(1)$ cuts. Given jVj nodes in the CEG, the total number of cuts that need to be simulated by the accelerated method is $O(jVj)$. It is much smaller than $O(2^{jVj})$, the number of cuts that need to be simulated by the basic method.

Algorithm 3: *FindApplyOptLACSet_Cut*, a function for finding and applying an optimized LAC set to reduce delay by the priority cut-based method.

Input: the current approximate circuit G_{apx}, an error upper bound e_b, and a given limit λ.
Output: a flag *HasSol*, denoting the existence of a valid LAC set, and a new approximate circuit G_{new}.

1 $G_{apx} \Leftarrow$ *PreprocessNegativeCapacity*(G_{apx});
// Get reference solution by maximum flow
2 $(HasSol_{ref}, G_{ref}) \Leftarrow$ *FindApplyOptLACSet_Flow*(G_{apx}, e_b);
3 **if** $HasSol_{ref} = true$ **then** error limit $\epsilon \Leftarrow$ *GetError*(G_{ref});
4 **else** error limit $\epsilon \Leftarrow e_b$;
// Improve solution by priority cut
5 $CEG \Leftarrow$ *BuildCriticalErrorGraph*(G_{apx});
6 Add a sink node t into *CEG*;
7 **foreach** *functional/sink node* $n \in$ *CEG in topo. order* **do**
8 | Get $\Psi(n)$ by Eq. (5) with parameters ϵ and λ;
9 Global cut $\mathcal{N} \Leftarrow$ the cut in $\Phi(t)\backslash\{t\}$ with the smallest error;
10 LAC set $\mathcal{L} \Leftarrow$ *GetMinErrorLACs*$(\mathcal{N})$;
11 **if** *GetError*$(\mathcal{L}) \leq e_b$ **then**
12 | $G_{new} \Leftarrow$ *ApplyLACSet*$(G_{apx}, \mathcal{L})$; **return** $(true, G_{new})$;
13 Local cut $\mathcal{N}^0 \Leftarrow$ the local cut with the smallest error from all $\Psi(n)$'s, where n is a functional node in *CEG*;
14 LAC set $\mathcal{L}^0 \Leftarrow$ *GetMinErrorLACs*$(\mathcal{N}^0)$;
15 **if** *GetError*$(\mathcal{L}^0) \leq e_b$ **then**
16 | $G_{new} \Leftarrow$ *ApplyLACSet*$(G_{apx}, \mathcal{L}^0)$; **return** $(true, G_{new})$;
17 $G_{new} \Leftarrow G_{apx}$; **return** $(false, G_{new})$;

3) Entire Flow of Priority Cut-based Method: Algorithm 3 shows the priority cut-based implementation of the function *FindApplyOptLACSet*, which finds an optimized set of LACs and applies them to simplify the approximate circuit. It also starts with a pre-processing of the input approximate circuit to avoid the occurrence of negative MEIs in the CEG (Line 1). After that, Line 2 obtains a reference solution by the maximum flow-based method to determine an important parameter, error limit , for efficient cut enumeration. If the reference solution exists, namely, the maximum flow-based method can find a set of LACs to reduce the circuit delay without violating the error bound e_b, then Line 3 sets as the error of the approximate circuit G_{ref} produced by the maximum flow-based method. Otherwise, Line 4 sets as e_b. Then, we try to improve the reference solution by efficiently enumerating priority cuts. To do this, Line 5 builds a CEG from the pre-processed approximate circuit G_{apx}, and Line 6 adds a sink node t to the CEG. Then, Line 7 traverses each functional/sink node in the CEG following a topological order, and a set of priority cuts of each node n, (n), is obtained by Eq. (5) based on the error limit and the parameter (Line 8). Next, Lines 9–10 derive the global cut with the smallest error from $(t)nftg$ and the set of min-error LACs L for all the nodes in the global cut. If the error of the LAC set L does not exceed the error bound e_b (Line 11), then Line 12 applies the LACs in L to further approximate the circuit G_{apx}, and returns *HasSol* = *true* and the resulting circuit G_{new}. Otherwise, when there is no global cut with error no more than e_b, we further explore the local cuts of all functional nodes in the CEG, since the LACs on local cuts may contribute to delay reduction too. Line 13 selects the best local cut N^0 with the smallest error from all (n)'s, where n is a functional node in the CEG. Line 14 obtains the set of min-error LACs L^0 for all nodes in N^0. If the error of the LAC set L^0 does not exceed the error bound e_b, then L^0 is used to simplify G_{apx}, and Line 16 returns *HasSol* = *true* and the resulting circuit G_{new}. However, it is also possible that there is no global or local cut with error no more than e_b. In this case, Line 17 returns *HasSol* = *false* and the pre-processed circuit G_{apx}.

Compared to the maximum flow-based method, the priority cut-based method has some computation overhead. By analyzing Algorithm 3, we find that the overhead mainly lies in the error evaluation of many local cuts by logic simulation, which helps build the set of priority cuts for each node (Lines 7–8 of Algorithm 3). As discussed at the end of Section VI-B2, the number of these local cuts is $O(jVj)$. Assume that the number of input patterns used in each logic simulation is M and the number of nodes in the entire circuit is W. Then, the time complexity to simulate the circuit to get the error of one cut is $O(MW)$. As the simulation is performed for $O(jVj)$ cuts to get their errors, the computation overhead of the priority cut-based method is $O(MWjVj)$.

VII. Experimental Results

This section presents the experimental results. All the experiments are carried out on a computer with an Intel Xeon Gold 6146 CPU (3.20GHz) and 256 GB memory running Ubuntu 20.04. HEDALS is implemented in C++ and tested with a single thread of the CPU. The average errors are measured through logic simulation. We assume that the input patterns are uniformly distributed, although other input distributions can also be handled. For each logic simulation, we randomly generate 100,000 input vectors, which are sufficient to obtain average errors with a high accuracy [9], [16]. The delay-oriented synthesis and mapping in HEDALS (*i.e.*, the function *SynthesizeAndMap* in Algorithm 1) are performed by ABC [20]. We apply the ABC script "*resyn*; *resyn2*" 6 times for technology-independent synthesis and the ABC command *map* for

technology mapping using the Nangate 45nm open cell library [21]. We also use ABC to report the circuit area in m^2 and delay in ns after mapping, where the delay is reported by the STA command *stime*. We use delay and area ratios to evaluate the cost of the approximate designs. The delay ratio (*resp.* area ratio) is defined as the ratio of the delay (*resp.* area) of the approximate circuit over that of the original one. Obviously, smaller delay and area ratios are preferred.

TABLE I. BENCHMARK CIRCUIT INFORMATION.

Benchmark suite	Circuit	#I/Os	AIG		Gate-netlist	
			Size	Depth	Area	Delay
ISCAS85	c880	60/26	313	22	231.7	0.38
	c1355	41/32	390	16	429.1	0.45
	c1908	33/25	367	25	378.8	0.61
	c2670	233/140	579	17	534.1	0.40
	c3540	50/22	937	32	716.3	0.78
	c5315	178/123	1306	28	951.0	0.57
	c7552	207/108	1469	26	1030.2	1.06
BACS	absdiff	16/9	104	14	101.9	0.30
	add32	64/33	302	20	254.0	0.43
	buttfly	32/34	227	31	212.0	0.50
	mac	12/8	124	20	142.8	0.37
	mult8	16/16	470	44	496.4	0.88
	mult16	32/32	2033	41	2337.9	0.84
EPFL	add128	256/129	1019	314	982.9	4.83
	barshift	135/128	2688	14	1945.0	0.85
	divisor	128/128	23667	4473	19949.5	89.78
	log2	32/32	38540	419	26422.8	11.56
	max	512/130	2686	549	2456.2	10.98
	mult64	128/128	33242	326	22401.5	6.87
	sine	24/25	7044	180	5334.4	4.50
	sqrt	128/64	21951	4591	19035.5	128.16
	square	64/128	20030	296	14394.6	5.88

Table I lists benchmarks used in the experiments, including several circuits from the ISCAS85 benchmark suite [19], all arithmetic circuits from the BACS benchmark suite [13], and some largest circuits from the EPFL benchmark suite [22]. Both the AIG and gate-netlist representations are considered, and they are optimized by ABC to fully minimize their delays. Specifically, to produce the AIG representation of a benchmark, we first convert the benchmark into AIG with the ABC command *strash* and then fully optimize the AIG by repeatedly applying the delay-oriented synthesis script *resyn2* until its quality cannot be further improved. The sizes and depths of the optimized AIGs are listed in columns 4 and 5 of Table I, respectively. To produce the gate-netlist representation of a benchmark, we first convert it into an AIG with *strash* and fully optimize the AIG delay in the same way as we do for the AIG representation. Then, we apply the delay-oriented technology mapping command *map* to the AIG. The last two columns of Table I list the areas and delays of the gate netlists, respectively.

In what follows, several sets of experiments are designed to study the performance of HEDALS. They involve various types of LACs, circuit representations, and error constraints.

A. Study under the NMED Constraint

This section performs experiments under the NMED constraint on the BACS benchmarks listed in Table I. NMED is a commonly-used error metric for arithmetic circuits. We apply HEDALS on the AIG representation of a circuit, but finally, we map the AIG into a gate netlist for area and delay evaluation.

1) Accuracy and Impact of the Linear Error Estimation Models: Section VI-A proposes a linear model, Eq. (3), to estimate the error of a LAC set. We first evaluate its accuracy on the adder and multipliers in the BACS benchmark suite, *i.e.*, *adder32*, *mult8*, and *mult16*. We run the maximum flow-based HEDALS using the linear error estimation model. The applied LAC is the constant LAC [3], *i.e.*, replacing a node by a constant 0 or 1. For each benchmark, we consider the first 7 iterations in the HEDALS flow. For each optimized LAC set found in each iteration, we compare its *estimated error* (*EER*) obtained by Eq. (3) and the *actual error* (*AER*) obtained by logic simulation. We use the same set of 100,000 input vectors to calculate the EER and AER. As shown in Fig. 5, except iterations 4 and 6 for *mult8* and iteration 7 for *mult16*, the EER is close to AER for all the 7 iterations of the three circuits.

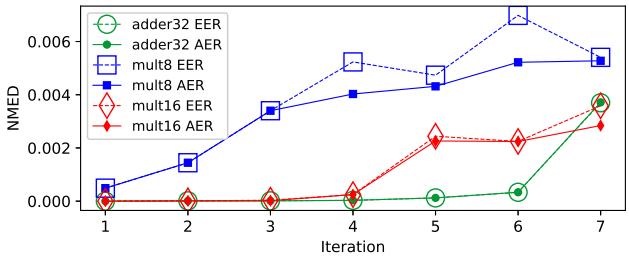


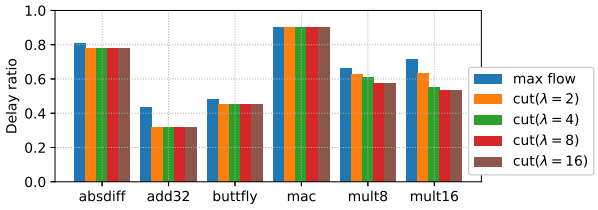
Fig. 5. Comparison of estimated error (EER) and actual error (AER). The error metric is NMED.

In the prior work [12], a less accurate linear error estimation model shown in Eq. (2) is proposed. We further compare the impacts of this model and our more accurate model based on Eq. (3). We consider each point in each AER curve in Fig. 5, which corresponds to an intermediate approximate design. We apply Eqs. (2) and (3), respectively, to each intermediate design and obtain their resulting optimized LAC sets. Table II lists all the circuits and their iterations where the LAC set found using Eq. (2) differs from that found using (3). It also lists the AERs of those different LAC sets. We can see that our model, Eq. (3), can lead to better LAC sets with smaller AERs. We also compare the final circuit quality achieved by both models under two NMED bounds, 0.3% and 3.0%. Compared to using Eq. (2), using Eq. (3) reduces more delay by 7.9%, 1.7%, and 0.8%, on average, for *adder32*, *mult8*, and *mult16*, respectively, while the average circuit area does not increase.

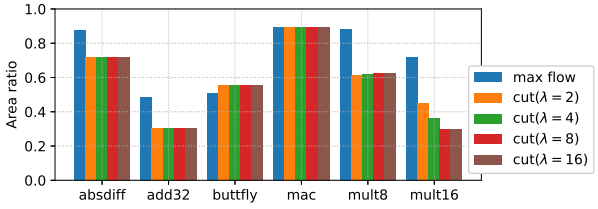
TABLE II. ALL DIFFERENT OPTIMIZED LAC SETS FOUND BY DIFFERENT LINEAR ERROR MODELS IN THE FIRST 7 ITERATIONS OF HEDALS. THE **BOLD** ENTRIES MEAN THAT EQ. (3) LEADS TO BETTER LAC SETS WITH SMALLER AERS THAN EQ. (2).

Circuit	Iteration	AER of optimized LAC set by Eq. (2)	AER of optimized LAC set by Eq. (3)
adder32	3	1.45E-05	**1.14E-05**
adder32	6	3.75E-04	**3.34E-04**
mult8	6	8.10E-03	**5.22E-03**
mult16	6	2.25E-03	**2.24E-03**

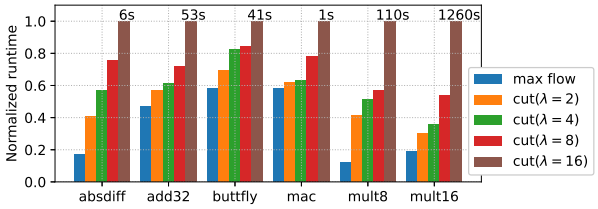
2) Comparing Maximum Flow-based Method with Priority Cut-based Method: In the HEDALS framework, a maximum flow-based method and a priority cut-based method are proposed. We compare their performance in this section with an NMED bound of 0.005. The applied LAC is the constant LAC.



(a) Delay comparison for various benchmarks and methods.



(b) Area comparison for various benchmarks and methods.



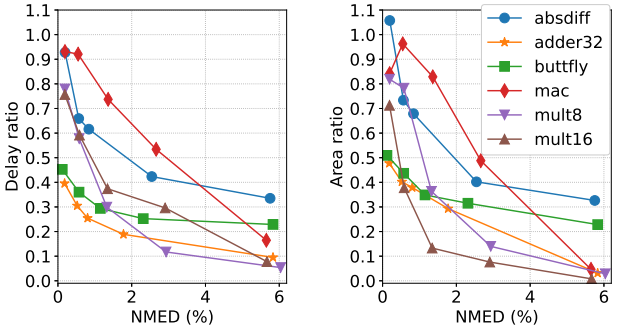
(c) Runtime comparison for various benchmarks and methods.

Fig. 6. Comparison between the maximum flow-based and priority cut-based methods on the BACS benchmark suite under an NMED bound of 0.005.

Fig. 6 compares the delay ratio, area ratio, and runtime of the maximum flow-based and priority cut-based methods. Different limits ’s are used in the priority cut-based method. For each circuit, the runtime is normalized to that of the priority cut-based method with = 16.

As shown in Figs. 6(a) and 6(b), the priority cut-based method reduces more delay and area than the maximum flow-based method on most benchmarks. The reason is that using priority cuts, we can find a set of LACs causing a smaller error in each iteration of the HEDALS flow. Then, the error increase of each iteration is smaller, leading to more iterations and hence, more sets of LACs applied to simplify the circuit. However, there are two exceptions. The first is *mac*. Its approximate designs generated by both methods have the same delay and area. Actually, in each iteration during the synthesis of *mac*, the set of LACs found by both methods are exactly the same, and hence, the final approximate circuits are identical. The second one is *buttfly*, for which the priority cut-based method produces approximate circuits with smaller delays but larger areas. In terms of runtime, the priority cut-based method takes a longer runtime, since it calls the maximum flow-based method to generate a reference solution.

In terms of the influence of the parameter , Fig. 6(a) shows that delay ratio decreases or stays the same as increases. As shown in Fig. 6(b), area ratio behaves similarly as delay ratio with different ’s. It is reasonable since with a larger , more priority cuts in the CEG are considered. Thus, it is more likely to find a better global cut with a smaller error. This leads to more approximation iterations and hence, more delay and area reduction. Furthermore, as changes from 8 to 16, the delay ratios of all circuits remain unchanged. It implies that by keeping at most 8 priority cuts for each node in the CEG, we can find good enough approximate circuits with small delays. In terms of efficiency, Fig. 6(c) shows that the runtime increases with . This is reasonable since with a larger , more priority cuts are considered for each node, which takes a longer runtime. Considering delay, area, and runtime, we can see that the priority cut-based method with = 8 can achieve good delay and area savings in a short time. Thus, in the remaining experiments, we choose this implementation for HEDALS unless otherwise specified.



(a) Delay-NMED tradeoff. (b) Area-NMED tradeoff.

Fig. 7. Quality-NMED tradeoff on the BACS benchmark suite.

3) Quality-NMED Tradeoff of Approximate Designs: This section studies the quality-NMED tradeoff of the approximate circuits synthesized by HEDALS. To show that HEDALS can support various LACs, we select another LAC, the ALSRAC LAC [7].

We run HEDALS on each BACS benchmark under 5 NMED bounds, *i.e.*, $\frac{1}{2^9\ 1}$ 0:20%, $\frac{3}{2^9\ 1}$ 0:59%, $\frac{7}{2^9\ 1}$ 1:37%, $\frac{15}{2^9\ 1}$ 2:94%, and $\frac{31}{2^9\ 1}$ 6:07%. Fig. 7(a) plots the delay ratio-NMED curves. For each circuit, the delay decreases monotonically with the NMED. When the NMED reaches 6:07% (see the rightmost points on the curves), the delay ratio of different circuits ranges from 5% (*i.e.*, *mult8*) to 34% (*i.e.*, *absdiff*). Although area saving is a byproduct of HEDALS, we also plot the area ratio-NMED curves in Fig. 7(b). From Fig. 7(b), areas of most approximate circuits are less than those of the exact counterparts. Moreover, the area decreases with the NMED for all the circuits except *mac*. For *mac*, when the NMED changes from 0:20% to 0:59%, its area increases. It is because HEDALS works on the AIG representation of a circuit in this experiment, but the area and delay are evaluated after technology mapping. As the NMED changes from 0.20% to 0.59% for *mac*, the AIG size keeps unchanged and the depth decreases, which is expected. However, its area increases after mapping due to an occasional inconsistency between the quality change trend of an AIG and that of a mapped gate netlist. When the NMED approaches 6:07% (see the rightmost points on the curves), the area ratio of different circuits varies from 1% (*i.e.*, *mult16*) to 33% (*i.e.*, *absdiff*).

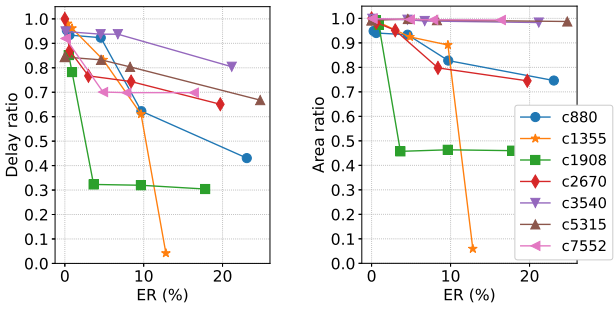
We also compare HEDALS with a state-of-the-art ALS flow, *BLASYS* [9], which is an open-source area-oriented flow. BLASYS is run with 48 threads of our CPU, and we report its runtime as the total runtime of all threads. For a more fair comparison with our delay-oriented HEDALS flow, BLASYS is slightly modified to enhance its ability in reducing delay. Specifically, the modified BLASYS applies the same delay-oriented synthesis and mapping

TABLE III. COMPARISON BETWEEN HEDALS AND BLASYS UNDER THE NMED CONSTRAINT. THE **BOLD** VALUES MEAN THAT HEDALS OUTPERFORMS BLASYS.

Circuit	NMED bound	Delay ratio HEDALS	Delay ratio BLASYS	Area ratio HEDALS	Area ratio BLASYS	Runtime/min HEDALS	Runtime/min BLASYS
absdiff	0.59%	**65.9%**	103.9%	**73.4%**	83.5%	**0.09**	2.84
	2.94%	**42.3%**	98.3%	**40.2%**	74.7%	**0.14**	6.09
adder32	0.59%	**30.5%**	43.6%	40.2%	31.9%	**0.54**	39.00
	2.94%	**18.9%**	24.3%	**29.3%**	33.1%	**0.55**	41.14
buttfly	0.59%	**36.0%**	45.7%	**43.7%**	61.7%	**0.29**	27.08
	2.94%	**25.3%**	37.5%	**31.5%**	41.2%	**0.41**	29.63
mac	0.59%	**92.1%**	97.6%	96.3%	95.2%	**0.05**	0.22
	2.94%	**53.4%**	104.8%	**48.8%**	72.4%	**0.14**	2.61
mult8	0.59%	**57.8%**	60.7%	78.2%	34.5%	**1.18**	1168.26
	2.94%	**11.7%**	34.0%	**14.0%**	16.2%	**2.10**	1235.36
mult16	0.59%	**59.2%**	70.6%	37.8%	15.8%	**21.59**	2835.34
	2.94%	**29.6%**	52.1%	**7.6%**	7.8%	**23.06**	3007.71
Average		**43.6%**	64.4%	**45.1%**	47.3%	**4.18**	699.61

process used in HEDALS, *i.e.*, applying the ABC script "resyn; resyn2" 6 times, followed by the ABC command *map*. We compare the approximate designs generated by HEDALS and BLASYS under two NMED bounds, 0.59% and 2.94%. The results are listed in Table III. We can see that for all benchmarks, the delay ratio of HEDALS is much smaller than that of BLASYS. On average, HEDALS further reduces the delay ratio by a relative value of 32.3% over BLASYS, which shows the effectiveness of HEDALS in delay optimization. Furthermore, we find that for all benchmarks under some NMED bounds, HEDALS can even reduce more area. The reason is that BLASYS partitions a circuit into sub-circuits and reduces the area of each sub-circuit separately. This method may miss some area reduction opportunities, such as the simplification across two sub-circuits. On the contrary, HEDALS does not partition the circuit and can consider some opportunities that BLASYS misses, hence reducing more area than BLASYS sometimes. As for efficiency, HEDALS is much faster than BLASYS on all benchmarks. On average, HEDALS accelerates by 167 over BLASYS. The acceleration arises from two aspects. First, although both HEDALS and BLASYS simplify circuits iteratively, HEDALS has fewer iterations. Its reason is that HEDALS applies multiple LACs in each iteration, while BLASYS only applies one. Hence, HEDALS modifies more sub-circuits in each iteration and approaches the error bound faster. Second, the runtime of HEDALS for each iteration is much shorter, since it only focuses on the nodes on the critical graph of the circuit and only considers the min-error LACs of these nodes. Besides, it prunes LAC sets with large errors by the priority cut-based method.

B. Study under ER Constraint



(a) Delay-ER tradeoff. (b) Area-ER tradeoff.

Fig. 8. Quality-ER tradeoff on the ISCAS benchmark suite.

This section studies the performance of HEDALS on the 7 ISCAS85 circuits listed in Table I under the ER constraint. ER measures the erroneous probability of a circuit. It is suitable for evaluating the accuracy of circuits such as classifiers, controllers, and error correctors (*e.g.*, *c1355* and *c1908*). It is also widely used as an additional error metric for arithmetic circuits (*e.g.*, *c7552*) [23] and arithmetic logic units (*e.g.*, *c880*, *c2670*, *c3540*, and *c5315*) [4]. To further show the wide applicability of HEDALS, we apply it to the gate-netlist representation of these benchmarks with the constant LAC.

We run HEDALS on each benchmark under 5 ER bounds, 0.5%, 1%, 5%, 10%, and 25%. Fig. 8(a) plots the delay ratio-ER curves of the approximate circuits generated by HEDALS. For all circuits, the delay decreases monotonically with the ER. When the ER reaches 25% (see the rightmost points on the curves), the delay ratio of different circuits ranges from 4% (*i.e.*, *c1355*) to 80% (*i.e.*, *c3540*). We also plot the area ratio-ER curves in Fig. 8(b). For all approximate circuits, their areas are smaller than the areas of the exact counterparts. Moreover, the area decreases monotonically with the ER for all circuits. For some circuits such as *c3540*, *c5315*, and *c7552*, the area is almost unchanged. However, for some other circuits such as *c1355* and *c1908*, the area also reduces by a large amount.

We also compare HEDALS with BLASYS, where the setup of BLASYS is the same as that in Section VII-A3. We compare the approximate designs generated by HEDALS and BLASYS under two ER bounds, 0.5% and 5%. The results are listed in Table IV. We can see that the delay ratio of HEDALS is always smaller than that of BLASYS except for the case of *c2670* under an ER bound of 0.5%. In this case, HEDALS cannot simplify *c2670*, while BLASYS can. On average, HEDALS further reduces the delay ratio by a relative value of 6.1% over BLASYS. Table IV also shows that HEDALS reduces less area than BLASYS, but it is guaranteed that the area of an approximate circuit produced by HEDALS is no larger than that of the exact counterpart. As for efficiency, HEDALS is much faster than BLASYS on all circuits. On average, HEDALS accelerates by 9799 over BLASYS.

TABLE IV. COMPARISON BETWEEN HEDALS AND BLASYS UNDER THE ER CONSTRAINT. THE **BOLD** VALUES MEAN THAT HEDALS IS BETTER THAN BLASYS.

Circuit	ER bound	Delay ratio HEDALS	Delay ratio BLASYS	Area ratio HEDALS	Area ratio BLASYS	Runtime/min HEDALS	Runtime/min BLASYS
c880	0.5%	**95.3%**	98.6%	94.8%	89.9%	**0.01**	24.1
	5.0%	**92.3%**	93.9%	93.3%	75.0%	**0.04**	48.6
c1355	0.5%	**98.1%**	120.8%	99.4%	98.3%	**0.03**	70.6
	5.0%	**83.3%**	94.1%	**92.4%**	95.5%	**0.05**	200.3
c1908	0.5%	**85.2%**	90.6%	99.3%	90.9%	**0.02**	55.5
	5.0%	**32.3%**	33.7%	45.7%	41.9%	**0.16**	385.4
c2670	0.5%	100.0%	79.6%	100.0%	64.5%	**0.03**	297.6
	5.0%	**76.7%**	81.5%	95.1%	63.7%	**0.17**	347.7
c3540	0.5%	**94.8%**	99.7%	99.8%	92.0%	**0.02**	445.4
	5.0%	**93.7%**	101.6%	99.4%	83.0%	**0.04**	1366.0
c5315	0.5%	**84.4%**	88.2%	99.2%	96.5%	**0.05**	2267.6
	5.0%	**83.2%**	86.9%	99.7%	97.5%	**0.15**	2538.6
c7552	0.5%	**92.0%**	93.7%	99.9%	84.4%	**0.07**	1362.9
	5.0%	**70.1%**	95.4%	99.5%	81.7%	**0.30**	1564.5
Average		**84.4%**	89.9%	94.1%	82.5%	**0.08**	783.9

C. Study under MHD Constraint

To show its scalability, we run HEDALS on the large EPFL benchmarks listed in Table I under the NMHD

constraint. We apply HEDALS on the AIG representation of a circuit, and the applied LAC is ALSRAC. Since these benchmarks are large, we apply the priority cut-based method with = 1 to save runtime. We also compare HEDALS with BLASYS, but we do not run the BLASYS program due to its long runtime. For example, as reported in [9], it takes BLASYS about 19 days to generate an approximate circuit for the benchmark *sine* with 7044 AIG nodes under an NMHD bound of 5%. Instead, we directly use the data from [9] for comparison. Although a different 65nm standard cell library is used in [9], the area and delay ratios from [9] are still good references.

We compare the results of HEDALS and BLASYS on the EPFL benchmarks under the same two NMHD bounds used in [9], 5% and 10%. As shown in Table V, the circuit delay is significantly reduced by HEDALS. On average, the delay ratio of HEDALS is 45.3%, which is about half of that of BLASYS. Particularly, HEDALS saves much delay on *divisor*. It reduces 97.7% and 99.2% delay under NMHD bounds of 5% and 10%, respectively. Meanwhile, the average area ratio of HEDALS is 69.2%, which is relatively 19.2% smaller than BLASYS. Furthermore, HEDALS is very efficient: its runtime for most designs is less than one hour.

TABLE V. PERFORMANCE OF HEDALS ON THE LARGE EPFL CIRCUITS. THE **BOLD** VALUES MEAN THAT HEDALS OUTPERFORMS BLASYS. DATA OF BLASYS ARE FROM [9]. N/A MEANS THAT THE CORRESPONDING DATA IS NOT REPORTED IN [9].

Circuit	NMHD bound	Delay ratio		Area ratio		Runtime/min	
		HEDALS	BLASYS	HEDALS	BLASYS	HEDALS	BLASYS
add128	5%	**35.1%**	90.8%	**81.3%**	89.4%	**0.3**	340.3
	10%	**15.6%**	80.9%	**72.6%**	79.4%	0.4	N/A
barshift	5%	**91.8%**	105.6%	**87.1%**	95.8%	**19.2**	3510
	10%	**83.4%**	88.5%	**84.0%**	90.0%	34.6	N/A
divisor	5%	**2.3%**	91.6%	**6.3%**	85.9%	62.8	N/A
	10%	**0.8%**	73.0%	**4.3%**	76.2%	89.3	N/A
log2	5%	**77.8%**	100.5%	96.8%	92.9%	19.1	N/A
	10%	**70.5%**	78.2%	96.6%	82.1%	53.0	N/A
max	5%	**21.1%**	114.3%	**34.5%**	91.0%	6.2	N/A
	10%	**20.5%**	94.3%	**31.1%**	77.6%	4.5	N/A
mult64	5%	**59.0%**	99.4%	88.8%	87.7%	14.7	N/A
	10%	**53.3%**	93.8%	90.2%	80.5%	21.2	N/A
sine	5%	**76.5%**	93.1%	94.9%	84.3%	**1.3**	27849.3
	10%	**70.6%**	79.9%	93.6%	71.7%	2.1	N/A
sqrt	5%	50.6%	N/A	52.4%	N/A	332.0	N/A
	10%	40.5%	N/A	42.1%	N/A	394.3	N/A
square	5%	**25.0%**	85.8%	97.9%	95.8%	14.5	N/A
	10%	**21.6%**	75.5%	90.3%	88.5%	19.4	N/A
Aver. w/o sqrt		**45.3%**	90.3%	**69.2%**	85.6%	60.5	N/A

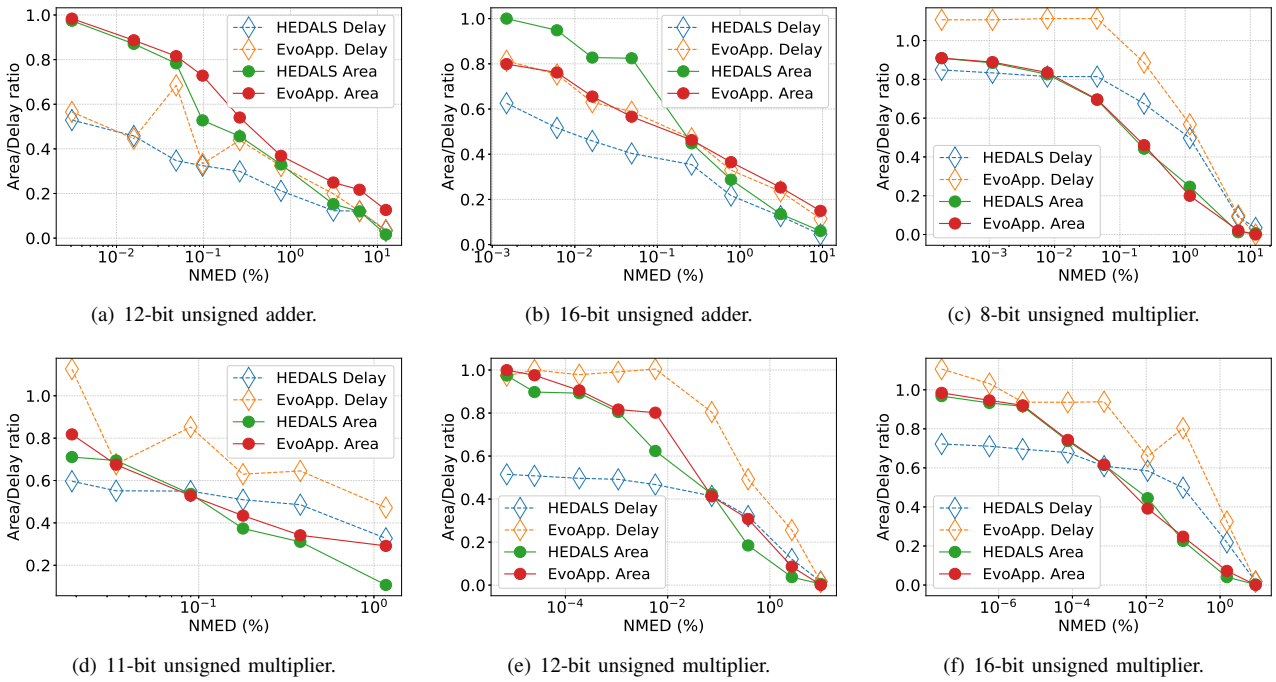
D. Study on Adders and Multipliers

In this section, we generate various approximate adders and multipliers by HEDALS and compare them with the circuits in *EvoApproxLib* (version 2022) [24], a widely-used library of approximate arithmetic circuits, which collects a series of approximate circuits from [25]–[28], produced by evolutionary algorithm-based ALS methods. The selected benchmarks are some largest adders and multipliers in EvoApproxLib, *i.e.*, 12-bit and 16-bit unsigned adders, and 8-bit, 11-bit, 12-bit, and 16-bit unsigned multipliers. For each benchmark, we select the approximate designs in EvoApproxLib generated under the NMED (called mean absolute error in EvoApproxLib) constraint. Since the published designs in EvoApproxLib are synthesized with a different standard cell library, we re-synthesize and map the EvoApproxLib designs into our Nangate 45nm library with the same delay-oriented synthesis and mapping process used in HEDALS. Then, to compare with each design in EvoApproxLib, we apply HEDALS to generate an approximate circuit with the NMED bound as the NMED of the EvoApproxLib design. HEDALS works on the AIG representation using the ALSRAC LAC. Since sometimes there is a large gap between the error of an approximate design C generated by HEDALS and the error bound, we further simplify the design C with the ALSRAC LAC until the error bound is reached. This post-processing can further reduce circuit area without increasing delay.

The results are shown in Fig. 9. Each sub-figure of Fig. 9 plots the delay ratio-NMED and area ratio-NMED curves of the approximate designs synthesized by HEDALS and those from the EvoApproxLib for a benchmark. We can see that HEDALS always reduces more delay than EvoApproxLib, except for two cases, the 12-bit adder under an NMED of 0.018% and the 8-bit multiplier under an NMED of 25%. Moreover, the delays of some EvoApproxLib circuits sometimes exceed those of the corresponding exact circuits, while HEDALS guarantees that the delay of an approximate circuit is always smaller than that of the corresponding exact design. It is not surprising because HEDALS is designed for delay optimization, while the EvoApproxLib method is not. Furthermore, HEDALS saves more area than EvoApproxLib on the 12-bit adder and achieves competitive area savings on the rest benchmarks. It is because the EvoApproxLib designs are generated by the evolutionary algorithm, which features randomness and may miss some area reduction opportunities, while HEDALS uses a different greedy strategy to select the LAC sets and may capture some area reduction opportunities missed by the evolutionary algorithm.

E. Comparison of HEDALS Performance with Different LAC Types and Circuit Representations

Since HEDALS can be applied with different LAC types and circuit representations, it is also interesting to study the performance of HEDALS with different combinations of LAC type and circuit representation. We perform this study in this section on two arithmetic circuits, *adder32* and *mult8*, and two error metrics, ER and NMED. The selected ER bounds are 0.005, 0.01, 0.05, and 0.1, and the selected NMED bounds are 0.005%, 0.01%, 0.05%, and 0.1%. For each ER or NMED bound, we consider constant and ALSRAC LACs, and AIG and gate-netlist representations. Since the ALSRAC LAC cannot work on gate netlists [7], only 3 combinations are studied, *i.e.*, *CONST+AIG*, *CONST+GATE*, and *ALSRAC+AIG*. The performance of HEDALS with the 3 combinations of LAC type and circuit representation is shown in Table VI. The delay and area ratios of each approximate circuit generated by HEDALS with each combination are reported. Comparing *CONST+AIG* with *CONST+GATE*, we can see that the former reduces more delay and area in most cases. One possible reason is that there are more nodes in an AIG than in a gate netlist. Hence, an AIG has more candidate LACs for circuit simplification, leading to better approximate designs. Comparing *ALSRAC+AIG* with *CONST+AIG*, we can see that the former outperforms the latter with smaller delay and area ratios in most cases. It is not surprising since



(a) 12-bit unsigned adder. (b) 16-bit unsigned adder. (c) 8-bit unsigned multiplier.

(d) 11-bit unsigned multiplier. (e) 12-bit unsigned multiplier. (f) 16-bit unsigned multiplier.

Fig. 9. Comparison between the approximate designs synthesized by HEDALS and those from the EvoApproxLib on delay and area ratios versus NMED.

TABLE VI. COMPARISON OF HEDALS PERFORMANCE WITH DIFFERENT LAC TYPES AND CIRCUIT REPRESENTATIONS UNDER ER AND NMED CONSTRAINTS. "DR" MEANS DELAY RATIO, AND "AR" MEANS AREA RATIO. THE BEST CHOICE AMONG THE THREE COMBINATIONS OF LAC TYPE AND CIRCUIT REPRESENTATION IS HIGHLIGHTED IN **BOLD**.

Circuit	LAC+representation	ER≤0.005		ER≤0.01		ER≤0.05		ER≤0.1		Average	
		DR	AR	DR	AR	DR	AR	DR	AR	DR	AR
adder32	CONST+AIG	**69.9%**	**96.4%**	69.9%	96.4%	**66.8%**	95.9%	65.7%	**93.6%**	68.1%	**95.6%**
	CONST+GATE	99.4%	96.5%	99.4%	96.5%	89.1%	95.5%	89.1%	95.5%	94.2%	96.0%
	ALSRAC+AIG	**69.9%**	**96.4%**	**66.2%**	**96.2%**	69.3%	**95.4%**	**61.1%**	95.2%	**66.6%**	95.8%
mult8	CONST+AIG	97.0%	99.2%	97.0%	99.2%	97.0%	99.2%	95.0%	**95.0%**	96.5%	98.2%
	CONST+GATE	100.0%	100.0%	100.0%	100.0%	99.6%	99.8%	94.6%	99.7%	98.6%	99.9%
	ALSRAC+AIG	**94.4%**	**99.0%**	**94.4%**	**99.0%**	**93.3%**	**86.7%**	**89.6%**	99.5%	**92.9%**	**96.0%**
Circuit	LAC+representation	NMED≤0.005%		NMED≤0.01%		NMED≤0.05%		NMED≤0.1%		Average	
		DR	AR	DR	AR	DR	AR	DR	AR	DR	AR
adder32	CONST+AIG	**62.7%**	**60.3%**	**59.8%**	**59.4%**	58.5%	47.2%	**47.6%**	**37.5%**	57.1%	**51.1%**
	CONST+GATE	70.8%	91.0%	60.1%	73.8%	57.1%	74.7%	57.1%	74.7%	61.2%	78.5%
	ALSRAC+AIG	**62.7%**	**60.3%**	**59.8%**	**59.4%**	**53.1%**	**45.9%**	52.5%	38.7%	**57.0%**	**51.1%**
mult8	CONST+AIG	97.0%	99.2%	97.0%	99.2%	91.8%	**93.8%**	82.7%	92.7%	92.1%	96.2%
	CONST+GATE	100.0%	100.0%	100.0%	100.0%	90.5%	95.3%	89.5%	94.3%	95.0%	97.4%
	ALSRAC+AIG	**95.5%**	**96.1%**	**96.2%**	**98.2%**	**90.0%**	96.9%	**80.5%**	**87.0%**	**90.6%**	**94.5%**

the ALSRAC LAC is based on signal resubstitution, which is a more fine-grained circuit simplification technique than constant replacement. To sum up, *ALSRAC+AIG* is the best choice for the two circuits.

VIII. CONCLUSION

In this work, we proposed HEDALS, a highly efficient delay-driven approximate logic synthesis framework. Its basic idea is to establish a critical graph of a target circuit and find an optimized LAC set based on the graph, which is then applied to shorten all the critical paths simultaneously. A maximum flow-based method and a priority cut-based method are proposed to find an optimized LAC set. The former is faster, while the latter can lead to better approximate designs. The experimental results on a wide range of benchmarks show that HEDALS outperforms the state-of-the-art ALS approaches. Furthermore, it supports various LACs, circuit representations, and average error metrics. Thus, HEDALS is a promising solution for synthesizing approximate circuits with minimized delays.

REFERENCES

[1] M. M. Waldrop, "The chips are down for moore's law," *Nature News*, vol. 530, no. 7589, p. 144, 2016.

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

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

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

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

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

[7] C. Meng, W. Qian, and A. Mishchenko, "ALSRAC: Approximate logic synthesis by resubstitution with approxi-