ICCAD: U: DALS: Delay-driven Approximate Logic Synthesis

Zhuangzhuang Zhou (Author), Weikang Qian (Research Advisor)
University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, Shanghai, China
{zhouzhuangzhuang, qianwk}@sjtu.edu.cn
Author ACM Student Member Number: 6299254; Category: Undergraduate Research

1 PROBLEM AND MOTIVATION

As modern VLSI designs encompass more complexity and transistor technology reaches nanoscale, it has been increasingly difficult to improve the performance and energy consumption of circuits by conventional design methods [18]. On the other hand, many recent applications, including image rendering, signal processing, speech recognition and machine learning, are error tolerant by their nature. Their error tolerance is caused by various reasons. For example, some of them are resilient to input noises. Some of them have outputs intended for human perception and can tolerate errors imperceptible to users. Others do not offer a unique answer and a variety of answers are acceptable. These classes of applications can tolerate inexact computation in substantial portions of their execution [3]. Under this circumstance, *approximate computing* was proposed as a novel circuit design paradigm [4]. Its basic idea is to modify the function of a target circuit without affecting its usability in its application. If the modification is proper, the resulting circuit will have smaller area, lower power consumption, and better performance than the original version.

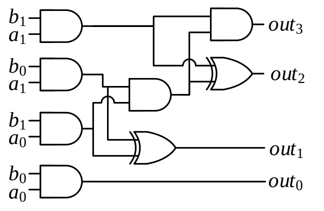
Fig. 1 shows the basic idea and effectiveness of approximate computing. For the 2-bit digital multiplier shown in Fig. 1b, we can introduce an error to its function by changing the only 4-bit output from "1001" to "111" [7]. The resulting Karnaugh-map for the approximate design is shown in Fig. 1c. The corresponding approximate circuit is shown in Fig 1d. We can see that for a 2-bit digital multiplier, the approximate design proposed in [7] is much simpler than the accurate one. It saves 3 gates and the number of basic gates lying on the critical path is reduced by 1. Thus, with the cost of 6.25% inaccuracy, the area and delay of the circuit are reduced by 37.5% and 33.3% respectively. Besides, the power consumption of the 2-bit multiplier circuit can be significantly reduced.

Given the large design space and difficulty in finding the most optimal way to inject error, a systematic methodology is needed to generate approximate designs for an arbitrary circuit. This is known as *approximate logic synthesis (ALS)*, which is the focus of our work. Given a target circuit, ALS seeks to synthesize an optimal approximate design that could satisfy the given error constraint, while trying to maximize the reduction of area, delay and power consumption of the circuit.

Significant progress has been made in ALS in recent years [1, 8, 9, 11, 13, 16, 17, 19–21]. However, all proposed ALS methods mainly focused on reducing the circuit area. Although the circuit delay is usually also reduced as a byproduct of these ALS methods, the potential power of ALS in improving circuit delay has not yet been fully explored. For error tolerantapplications such as real-time signal processing, they are , but they also have a stringent deadline to meet. For these applications, delay, instead of area, is the primary concern. Thus, if we can develop a delay-oriented ALS flow, it will be very helpful in synthesizing circuits for these applications.

a_1a_0 \ b_1b_0	00	01	11	10
00	0000	0000	0000	0000
01	0000	0001	0011	0010
11	0000	0011	1001	0110
10	0000	0010	0110	0100

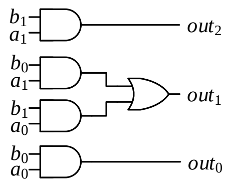
(a) Karnaugh map of an accurate multiplier.



(b) Circuit of the accurate multiplier.

a_1a_0 \ b_1b_0	00	01	11	10
00	000	000	000	000
01	000	001	011	010
11	000	011	111	110
10	000	010	110	100

(c) Karnaugh map of an approximate multiplier



(d) Circuit of the approximate multiplier.

Figure 1: Accurate and approximate designs of 2-bit multiplier.

Given the difficulty in optimizing the approximate circuits globally, previous ALS methods usually repeatedly apply approximate local changes (ALCs) to the gates in a circuit until the given error constraint is reached. In these approaches, where area is the primary concern, all gates in the circuit are treated equally since they contribute to the area equally regardless of their locations in the circuit. However, this is completely different for delay minimization. Circuit delay is determined by the critical paths and only the gates on the critical paths contribute to the delay. Furthermore, even doing local changes repeatedly on specific gates may not be effective. This is because there usually exist multiple critical paths of the same lengths. The local change may reduce the length of a particular critical path, but the lengths of the other critical paths still remain the same. Thus, to effectively reduce the delay, all the critical paths should be shortened simultaneously.

In this work, we propose DALS, a Delay-driven Approximate Logic Synthesis framework for delay-oriented ALS. DALS is a general framework that supports a wide range of ALCs and error metrics. One specific challenge in the DALS workflow is the selection of the sets of nodes to apply the ALCs, since there exist an exponential number of choices in a circuit. To solve this problem, we propose to establish a *critical error network (CEN)* from the circuit and then solve a maximal flow problem on the CEN. Our experimental results show that DALS produces approximate circuits with significantly reduced delays compared to the state-of-the-art ALS approaches.

2 BACKGROUND AND RELATED WORK

2.1 AND-Inverter Graph

An *AND-Inverter Graph (AIG)* is a directed acyclic graph (DAG) that implements a logic function [10]. Each node in an AIG is either a primary input (PI) or a two-input AND gate. If a node corresponds to a two-input AND gate, we call it a *functional node*. The nodes in the AIG that give the final outputs of the logic function are also marked as primary outputs (POs) of the AIG. The edges in an AIG can be complemented, indicating the inversion of the signal.

2.2 Error Metrics

There are two typical quantities to evaluate the error of an approximate circuit, *error rate (ER)* and *mean error distance (MED)*. ER is defined as the probability of an input pattern that gives an erroneous output for the approximate circuit. MED treats the output as a binary number and measures the average numerical deviation of the output of the approximate circuit from the correct output. For the approximate multiplier shown in Fig. 1d, if the inputs are uniformly distributed, its ER is $1/16 = 6.25\%$ and its MED is $2 \cdot 1/16 = 0.125$.

2.3 Related Works

Previous works have proposed a number of ALS methods [1, 8, 9, 11, 13, 16, 17, 19–21]. Since ALS is a computationally hard optimization problem, many state-of-the-art ALS techniques are based on applying ALCs to the circuits. For example, Shin and Gupta proposed to apply constant-0 and constant-1 replacement to internal gates [13]. Venkataramani *et al.* proposed to substitute one signal in the circuit by another with similar functionality [16]. Yao *et al.* proposed to perform local approximate disjoint bi-decomposition to internal signals to reduce the local area [21]. These previous works mainly focus on area reduction, with delay reduction as a side effect. The potential power of ALS in circuit delay reduction has not been fully explored.

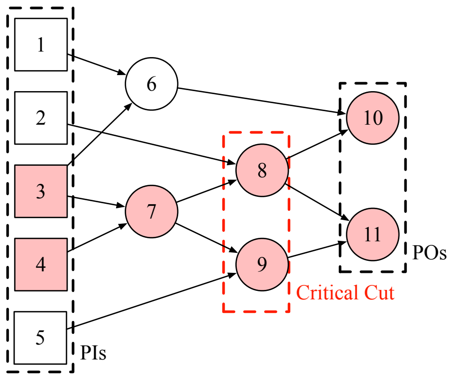
3 APPROACH AND UNIQUENESS

3.1 Methodology

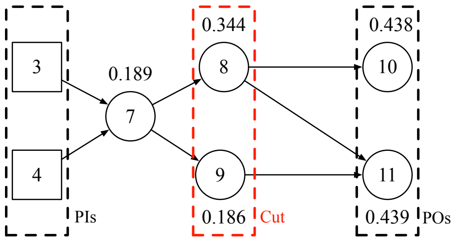
In this work, DALS is implemented on the AND-inverter graph (AIG) representation of a circuit [10]. The *critical graph* $G = (V, E)$ for an AIG is a subgraph of the AIG, where V and E are the sets of nodes and edges, respectively, on the critical paths of the AIG. For example, the colored nodes in Fig. 2a mark the critical graph of the AIG, which is shown in Fig. 2b.

We call a cut for a critical graph of an AIG a *critical cut* for the AIG. For example, nodes 8 and 9 form a critical cut for the AIG shown in Fig. 2a. The depth of the entire AIG can be reduced by reducing the depth of all nodes in the critical cut. In approximate computing, the depth of a node can be reduced by performing an ALC that could introduce error. There usually exist multiple critical cuts and multiple candidate ALCs for each node in the AIG. Different choices of the critical cut and ALCs could lead to different error impacts on the resulting approximate AIG.

In order to reduce the depth of an AIG, we need to shorten all critical paths simultaneously. Specifically, for a critical cut, if we can reduce the depth of all nodes in the cut, then the depth of the entire AIG can be reduced. For example, suppose that we are able



(a)



(b)

Figure 2: Illustration of (a) an AIG and (b) its critical graph.

to reduce the depths of nodes 8 and 9 in Fig. 2a both from 2 to 1, then the depth of the entire AIG reduces from 3 to 2.

In our approach, we will repeatedly select a critical cut and a set of ALCs to modify the nodes in the selected cut until the approximate AIG reaches the error constraint. Each iteration is expected to reduce the depth of the AIG by one, but at the same time, increase the error of the approximate AIG. Since we want to maximize the depth reduction, we should maximize the number of iterations. This requires that in each iteration, we should select the cut and the associated set of ALCs that lead to the minimum error impact among all choices. For simplicity, we call the cut *the best critical cut*.

The overall flow of DALS is summarized in Algorithm 1. To make the flow general, besides the input AIG and the error threshold, the flow also takes a user-specified approximation operator *op* as an input. The operator *op* should produce a set of ALCs for a node that could reduce the depth of the node. An example is replacing a node with a constant [13]. It includes two specific changes, replacement by a constant 0 and replacement by a constant 1. It can be easily seen that a constant replacement reduces the depth of the node to 0. Some other examples include the approximate substitution proposed in [16] and the approximate disjoint bi-decomposition proposed in [21].

3.2 Selecting the Best Critical Cut and the Associated Approximate Logic Changes

A crucial step (i.e., Line 5 in Algorithm 1) in our proposed flow is to find the critical cut and the associated set of ALCs that lead to the minimum error impact. For simplicity, we call the cut *the best*

Algorithm 1: The proposed flow of DALS.

```
Input: an AIG $G$, an error threshold $T$, and an approximation
       operator $op$.
Output: an approximate AIG $G_{apx}$ with reduced depth and error
        $e \le T$.
1 $G_{new} \Leftarrow G$;
2 while $e \le T$ do
3     $G_{apx} \Leftarrow G_{new}$;
4     $g \Leftarrow GetCriticalGraph(G_{apx})$;
5     $(Cut, ApxChange) \Leftarrow GetCutALC(g, op)$;
6     $G_{new} \Leftarrow ApplyChange(G_{apx}, Cut, ApxChange)$;
7     $e \Leftarrow GetError(G, G_{new})$;
8 return $G_{apx}$;
```

critical cut. The most straightforward approach is to enumerate all critical cuts and all sets of ALCs for each cut and then choose the combination with the minimum error. However, the total number of the critical cuts grows exponentially with the size of the AIG. Furthermore, for each cut, the number of applicable ALCs grows exponentially with the size of the cut. Bare enumeration is impractical for large AIGs. To solve this issue, we propose to transform this problem into a network flow problem. This transformation relies on our proposed estimation of the error impact of applying a set of ALCs to a critical cut.

3.2.1 *Estimating the Error Impact of Applying a Set of ALCs to a Critical Cut.*

The most straightforward way to evaluate the error impact of applying a set of ALCs to the critical cut is to apply the set of ALCs and then calculate the corresponding error metrics with simulation. However, the total number of critical cuts grows exponentially with the size of the AIG. And trying out all the available options with bare enumeration is impractical for large circuits.

To effectively estimate the error impact, we propose the following way to estimate the error impact of a set ALCs. Suppose that the critical cut contains m nodes $n_1, n_2, \ldots, n_m$ and for each $1 \le i \le m$, the ALC applied to node n_i is A_i. For each node n_i, we evaluate the error impact of applying ALC A_i to node n_i alone and denote the value as e_i. Then, we estimate the error impact of applying the set of ALCs to the nodes in the critical cut as the sum $e_1 + e_2 + \cdots + e_m$.

With error impact decomposition, we only need to keep the ALC that gives the minimum error impact for each functional node. We call this ALC the *optimal ALC* of the node and its error impact the *minimum error impact (MEI)* of the node. After obtaining the MEI of each functional node in the critical graph, we assign that value to the node. For example, the value near each functional node shown in Fig. 2b denotes the MEI of the node. Now, the problem of selecting the best critical cut simply becomes selecting a cut for the critical graph so that the sum of the MEIs of all nodes in the cut is minimal.

3.2.2 *Selecting the Best Critical Cut.*

In this section, we present a method to select the best critical cut. Our method first maps the original critical graph into a *critical error network (CEN)* and then solves a network flow problem on the CEN.

The CEN is built from the critical graph. We also need to assign proper capacities to the edges in CEN. The details for building the CEN is shown below.

(1) For each functional node n with MEI e in the critical graph, we add a pair of nodes n_a and n_b to the CEN. We also add an edge from n_a to n_b with capacity of e to the CEN.
(2) For each edge from a functional node u to a functional node v in the critical graph, we add an edge from u_b to v_a with infinite capacity to the CEN.
(3) We add a source node s. For each edge from a PI node p to a functional node n in the critical graph, we add an edge from s to n_a with infinite capacity to the CEN.
(4) We add a sink node t. For each PO node q in the critical graph, if it is a functional node, then we add an edge from q_b to t with infinite capacity to the CEN.

The CEN built from the critical graph shown in Fig. 2b is given in Fig. 3. The CEN is a classic flow network [2]. By the max-flow min-cut theorem [2], we can find a minimum cut of the CEN by solving the maximum flow problem on the CEN. Once we have identified each edge in the minimum cut, we can get the corresponding nodes in the critical graph from the mapping relation and obtain the cut in the critical graph that gives the minimum sum of MEIs.

$$\begin{tikzcd}
 & & 8_a \arrow[r, "0.344"] & 8_b \arrow[r, "\infty"] \arrow[ddr, "\infty"] & 10_a \arrow[r, "0.438"] & 10_b \arrow[dr, "\infty"] & \\
s \arrow[r, "\infty"] & 7_a \arrow[r, "0.189"] & 7_b \arrow[u, "\infty"] \arrow[d, "\infty"] & & & & t \\
 & & 9_a \arrow[r, "0.186"] & 9_b \arrow[r, "\infty"] & 11_a \arrow[r, "0.439"] & 11_b \arrow[ur, "\infty"] &
\end{tikzcd}$$

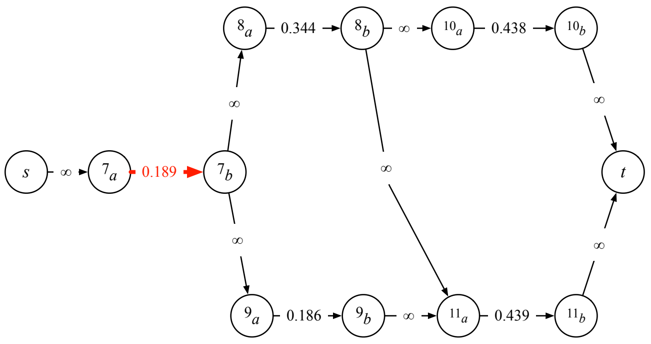


Figure 3: The critical error network built from the critical graph shown in Fig. 2b.

3.2.3 *The Entire Flow.*

Algorithm 2 shows the entire flow for finding the best critical cut and the associated set of ALCs,

3.3 Uniqueness

The novelty of the proposed approach is summarized as follows:

(1) Provide a general delay-driven ALS framework that can effectively reduce global delay.
(2) Build critical error network and formulate a max flow problem to find the best critical cut.
(3) Support a wide range of approximate local changes and some commonly-used error metrics.

4 RESULTS AND CONTRIBUTIONS

We performed two sets of experiments to evaluate the effectiveness of our DALS flow. First, we applied our method to some common benchmarks, and compared the result with the state-of-the-art area-driven approximation technique proposed in Su [14]. Second, we

Algorithm 2: The flow of the function *GetCutALC* for finding the best critical cut and the associated set of ALCs.

```
   Input: a critical graph $g$ and an approximation operator $op$.
   Output: the critical cut $Cut$ and the associated set of ALCs
           $ApxChange$ that lead to the minimum error impact.
1  foreach node $n$ in graph $g$ do
2      foreach ALC $x$ of node $n$ generated by the approximation operator
         $op$ do
3          obtain the error impact of applying ALC $x$ to node $n$;
4      $n.e \Leftarrow$ the minimum error impact over all ALCs of $n$;
5      $n.OptmALC \Leftarrow$ the ALC of $n$ with the minimum error impact;
6  $f \Leftarrow BuildCEN(g)$;
7  $minCut \Leftarrow SolveMaxFlow(f)$;
8  $Cut \Leftarrow EdgetoNode(minCut)$;
9  $ApxChange \Leftarrow GetBestALC(Cut)$;
10 return $Cut$ and $ApxChange$;
```

applied our method to synthesize approximate adders and compared the results with the state-of-the-art manual designs of approximate adders. We also compared the qualities of the approximate adders generated with our approach, and the ones generated with the method proposed by Chandrasekharan [1].

4.1 Study on Common Benchmarks under Error Rate Constraint

In this set of experiments, we selected several ISCAS85 benchmarks and an arithmetic circuit ALU4 synthesized by Synopsys Design Compiler [15]. We chose ER as the error metric.

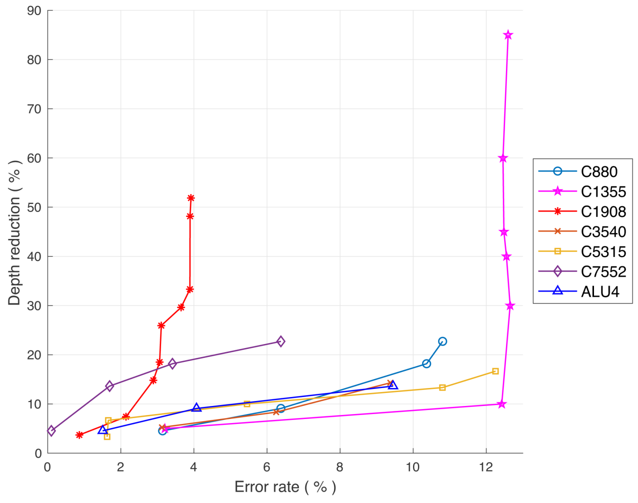


Figure 4: Depth reduction rate versus error rate by DALS.

The experimental results are shown in Fig. 4, which illustrates the relationship between depth reduction rate and ER for all the benchmarks. We can see that DALS can reduce the depth of the AIG when some inaccuracy is allowed and the depth reduction rate increases with ER.

We further took one point on the depth-reduction-rate-versus-ER curve for each benchmark and performed technology mapping to the approximate AIG to obtain the area and delay of the final mapped circuit. We can see that for all benchmarks, DALS can reduce the delay of the final mapped circuit when some inaccuracy is allowed. For benchmarks C880, C1355, and C1908, the circuit delays can be reduced dramatically compared to the introduced ER.

Table 1: DALS results and comparison with a state-of-the-art area-driven ALS method [14].

		DALS		[14][‡]	
circuit	error rate	Δarea[†]	Δdelay[†]	Δarea	Δdelay
C880	10.73%	17.90%	33.33%	24.04%	16.67%
C1355	12.48%	95.83%	93.83%	41.68%	2.53%
C1908	3.78%	58.24%	55.60%	58.63%	45.14%
C3540	14.31%	19.80%	16.37%	35.67%	8.33%
C5315	15.98%	3.01%	19.92%	13.10%	0.90%
C7552	6.38%	4.43%	16.90%	21.79%	0.91%
ALU4	9.45%	33.86%	19.23%	68.67%	7.69%

[†] ΔArea and ΔDelay are the area reduction rate and delay reduction rate, respectively, of the approximate circuit with respect to the original circuit.

[‡] The results are post-processed by delay-driven traditional logic synthesis.

To study the effectiveness of DALS, we compared it with a state-of-the-art area-driven ALS approach [14]. Previous area-driven ALS approaches have been proved to be effective in reducing circuit area. However, they do not focus on the delay. To make it fair, after a circuit had been synthesized by the ALS method in [14], we further applied delay-driven traditional logic synthesis from ABC to minimize the delay of the approximate circuit. The results of the comparison study are shown in the last two columns in Table 1.

From the results, we can see that even after subsequent delay-driven traditional logic synthesis, the state-of-the-art area-driven ALS method is not optimal in reducing delay. In certain cases, although the areas of the circuits reduce significantly, the delays still stay nearly the same. This is reasonable because an area-driven ALS method does not specifically optimize the nodes on the critical paths and it may choose other nodes to perform approximate changes. Thus, DALS can provide a better solution when delay is the primary goal. Since area reduction is just a side effect of DALS, it may not be as much as that of the area-driven ALS method. However, for benchmark C1355, DALS can even reduce more area.

4.2 Study on Approximate Adders under Mean Error Distance Constraint

In this set of experiments, we applied DALS to two accurate adder designs RCA_N8 and RCA_N16, which are 8-bit and 16-bit ripple carry adders, respectively, to generate the corresponding 8-bit and 16-bit approximate adders. We used MED, which is one of the most widely-used error metrics for approximate adders [5], as the error metric in DALS.

The experimental results are shown in Table 2. The synthesized adders were compared to two types of previously proposed manually designed approximate adders. They are *generic accuracy configurable adder (GeAr)* [12] and *accuracy-configurable adder (ACA)* [6]. The experimental results show that DALS can generate highly competitive approximate adders with reduced delays. In most cases, the adders synthesized by DALS have better areas, delays, MEDs, and MREDs than the previous manual designs at the cost of higher ERs. Thus, DALS can provide a better solution for many real-world applications where MED and MRED are more important than ER, such as image processing and machine learning. The fact that DALS generates adders with smaller MEDs but larger ERs is because the error metric is set as MED in DALS. Given this error metric, DALS tends to approximate the logic that affects the less significant outputs of the adder and it ignores the influence to ER.

Table 2: Approximate adders synthesized by DALS and comparison with some manually designed approximate adders.

circuit	ER	MRED	MED	area	delay
RCA_N8	0.00%	0.0000%	0.00	140	10.2
GeAr_N8_R2_P4	2.37%	0.6924%	1.50	138	8.6
DALS_N8_1	22.64%	0.5638%	1.07	134	8.4
GeAr_N8_R2_P2	18.73%	3.674%	7.52	128	7.0
DALS_N8_2	39.47%	2.804%	5.45	131	6.6
GeAr_N8_R1_P2	30.05%	7.104%	15.29	124	5.4
DALS_N8_3	69.92%	6.067%	13.61	85	5.4
RCA_N16	0.00%	0.0000%	0.00	315	13.4
GeAr_N16_R4_P4	5.86%	0.2657%	124.4	299	10.2
DALS_N16_1	51.85%	0.2321%	115.9	280	10.0
GeAr_N16_R2_P4	11.65%	0.9819%	510.4	290	8.6
DALS_N16_2	67.31%	1.0752%	514.6	269	8.2
ACA_II_N16_Q4	48.16%	3.893%	2049	260	7.0
DALS_N16_3	87.70%	3.024%	2043	207	7.0

Finally, we compared DALS to a previous ALS flow [1] that was also tested on approximate adders. The ALS flow [1] is based on approximation-aware rewriting of AIGs. We compared the quality of the 16-bit approximate adders synthesized by DALS to that by the previous flow [1]. The comparison results are shown in Table 3. To make it fair, we used the same 16-bit ripple carry adder design and the same setup as the experiments in [1] when testing DALS.

Table 3: Comparison between DALS and the work in [1] on the synthesis of 16-bit approximate adders.

circuit	ER/%	WCE	MBF	area	delay	area×delay	runtime/s
appx9	99.80	2038	9	254	13.4	3403.6	229
appx11	96.88	496	5	277	13.4	3711.8	201
appx13	99.22	1024	7	264	13.4	3537.6	220
DALS_N16_1	51.85	340	7	280	10.0	2800.0	6
appx12	99.90	4090	11	226	12.7	2870.2	187
DALS_N16_2	67.31	5380	11	269	8.2	2205.8	16
appx8	99.64	8320	13	120	7.0	840.0	151
appx10	99.64	8320	13	120	7.0	840.0	150
DALS_N16_3	87.70	6144	9	207	7.0	1449.0	23

Instead of MED, the previous work [1] chose to use the worst-case error (WCE) and maximum bit-flip (MBF) error to evaluate the quality of the approximate adders. From the table, we can see that the adder synthesized by DALS outperforms the corresponding approximated adder(s) from [1] in at least two error metrics among the three, i.e., ER, WCE, and MBF. Moreover, the DALS adder is much better in terms of area-delay product (ADP), which is a more comprehensive measure on the hardware quality. In conclusion, the approximate adders synthesized by DALS are better than those synthesized by the ALS method in [1] when the accuracy requirement is high.

4.3 Contributions

In this work, we proposed DALS, a delay-driven approximate logic synthesis framework, which can produce approximate circuits with significantly reduced delays. Its basic idea is to establish a critical error network (CEN) for the AIG representation of a target circuit and utilize the CEN to select the optimal set of nodes to apply depth-reduction approximate local changes. Experimental results show that DALS outperforms the state-of-the-art ALS approaches, and proves to be a promising solution for delay-oriented tasks.

5 PUBLICATION

[1] Z. Zhou, Y. Yao, S. Huang, S. Su, C. Meng, and W. Qian. 2018. DALS: Delay-driven Approximate Logic Synthesis. In ICCAD.

REFERENCES

[1] A. Chandrasekharan, M. Soeken, et al. 2016. Approximation-aware rewriting of AIGs for error tolerant applications. In *ICCAD*. 83:1–83:8.
[2] T.H. Cormen, C.E. Leiserson, et al. 2001. *Introduction to algorithms.* MIT Press.
[3] H. Esmaeilzadeh, L. Ceze A. Sampson, and D. Burger. 2012. Neural Acceleration for General-Purpose Approximate Programs. In *MICRO*. 449–460.
[4] J. Han and M. Orshansky. 2013. Approximate computing: An emerging paradigm for energy-efficient design. In *ETS*. 1–6.
[5] H. Jiang, J. Han, and F. Lombardi. 2015. A comparative review and evaluation of approximate adders. In *GLSVLSI*. 343–348.
[6] A.B. Kahng and S. Kang. 2012. Accuracy-configurable adder for approximate arithmetic designs. In *DAC*. 820–825.
[7] P. Kulkarni, P. Gupta, and M. Ercegovac. 2011. Trading Accuracy for Power with an Underdesigned Multiplier Architecture. In *VLSID*. 346–351.
[8] G. Liu and Z. Zhang. 2017. Statistically certified approximate logic synthesis. In *ICCAD*. 344–351.
[9] J. Miao, A. Gerstlauer, and M. Orshansky. 2014. Multi-Level approximate logic synthesis under general error constraints. In *ICCAD*. 504–510.
[10] A. Mishchenko, S. Chatterjee, and R. Brayton. 2006. DAG-aware AIG rewriting: A fresh look at combinational logic synthesis. In *DAC*. 532–535.
[11] A. Ranjan, A. Raha, et al. 2014. ASLAN: Synthesis of approximate sequential circuits. In *DATE*. 364:1–364:6.
[12] M. Shafique, W. Ahmad, et al. 2015. A low latency generic accuracy configurable adder. In *DAC*. 86:1–86:6.
[13] D. Shin and S. K. Gupta. 2011. A new circuit simplification method for error tolerant applications. In *DATE*. 1–6.
[14] S. Su, Y. Wu, and W. Qian. 2018. Efficient batch statistical error estimation for iterative multi-level approximate logic synthesis. In *DAC*. 54:1–54:6.
[15] Synopsys Inc. 2018. http://www.synopsys.com. (2018).
[16] S. Venkataramani, K. Roy, and A. Raghunathan. 2013. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In *DATE*. 1367–1372.
[17] S. Venkataramani, A. Sabne, et al. 2012. SALSA: Systematic logic synthesis of approximate circuits. In *DAC*. 796–801.
[18] M. M. Waldrop. 2016. The chips are down for Moore's law. *Nature* 530, 7589 (2016), 144–147.
[19] Y. Wu and W. Qian. 2016. An efficient method for multi-level approximate logic synthesis under error rate constraint. In *DAC*. 128:1–128:6.
[20] Y. Wu, C. Shen, et al. 2017. Approximate logic synthesis for FPGA by wire removal and local function change. In *ASPDAC*. 163–169.
[21] Y. Yao, S. Huang, et al. 2017. Approximate disjoint bi-decomposition and its application to approximate logic synthesis. In *ICCD*. 517–524.