Approximate Logic Synthesis by Multi-Objective Simulated Annealing

Yi-Che, Chen
B09901047
Dept. of Electrical Engineering
National Taiwan University

Che-Ming, Chang
B09901156
Dept. of Electrical Engineering
National Taiwan University

Yu, Chien
B09901189
Dept. of Electrical Engineering
National Taiwan University

***Abstract*—We proposed a method for Approximate Logic Synthesis (ALS) by applying the local approximate change (LACs) with a Multi-Objective Simulated Annealing (MOSA) engine. Our method includes LACs which can reduce the error rate with probability. With an alternation between LACs and exact circuit optimizations, we can explore better solutions compared to the existing ALS tool ALSRAC [1] on EPFL benchmarks.**

1. Introduction

Approximate computing is an emerging design paradigm that adds a further dimension to the design space with inaccuracy, which offers applications in diverse fields that are error-tolerant, including image processing, machine learning, and signal processing. *Approximate logic synthesis* (ALS) is the central issue in approximate computing, where the target is to automatically generate the approximate circuits within the given tolerated error bound. Circuits realized with ALS can offer significant performance increases, such as area and power reduction, concerning their exact counterparts [2].

Previous works have already proposed various ALS paradigms. Two-level approaches [3] aim at finding the approximate sum-of-product (SOP) form by finding the minimum number of 0-to-1 flips to minimize the literal count. Another popular paradigm is ALS on multi-level circuits, which usually consist of two phases. First *local approximate changes* (LACs) is applied on the current network. The quality of the result (QoR) is evaluated to estimate how much that transformation will impact the final result [2]. Greedily choosing the best LACs has shown good synthesis quality in multi-level Boolean networks [4, 1], yet they face the following challenges. This ALS paradigm requires an accurate evaluation of *multiple* error rate after applying different LACs in one iteration, which could be time-consuming and lead to runtime blowup. This motivates us to devise a tool that produces excellent ALS quality while maintaining the approach practical and scalable.

Our main contributions are as follows:

- We formalized the ALS problem into a constrained optimization problem, which can be solved by multi-objective simulated annealing (MOSA) with great performance.
- We introduce LACs to lower the error rate in exchange for area in simulated annealing. This is crucial for MOSA to reach a feasible solution without violating the error constraint.

1.1. Error Metrics

Error metrics are essential for measuring the accuracy of an approximate circuit. In a circuit with N primary inputs and M primary outputs, let the set of all possible input combinations be $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{2^N}\}$. We assume that the random vector chosen from the input space follows a uniform distribution, i.e., each input pattern occurs with an equal probability. Let $\mathbf{y}(\mathbf{x})$ and $\hat{\mathbf{y}}(\mathbf{x})$ be the outputs of the original circuit and approximate circuit for random input vector $\mathbf{x}$ respectively.

For **error rate (ER)**, the outputs will be viewed as incorrect if one of the primary is incorrect since one is usually of interest to know how often errors occur in approximate circuits, regardless of their magnitude. ER is mainly used to evaluate the accuracy of random / control circuits:

$$ER = \frac{1}{2^N} \sum_{\mathbf{x} \in \mathbb{B}^N} \mathbb{I}[\mathbf{y}(\mathbf{x}) \neq \hat{\mathbf{y}}(\mathbf{x})] \tag{1}$$

For **Hamming distance (HD)** (or termed *error magnitude* [5]), we will first calculate the Hamming distance of $\mathbf{y}(\mathbf{x})$ and $\mathbf{y}(\mathbf{x})$and divide it by M for normalization:

$$HD = \frac{1}{2^N} \sum_{\mathbf{x} \in \mathbb{B}^N} \frac{1}{M} d_H[\mathbf{y}(\mathbf{x}), \hat{\mathbf{y}}(\mathbf{x})] \tag{2}$$

1.2. Problem Formulation

Given an input combinational circuit $\mathbf{y}(\mathbf{x})$, output an approximate circuit $\hat{\mathbf{y}}(\mathbf{x})$ such that the mapped area is minimized while satisfying the error threshold constraint,

2. Methodology

2.1. Overall ALS Flow

Figure 1 shows the proposed ALS flow. Instead of choosing the best LAC in each iteration, we adopt fast simulations

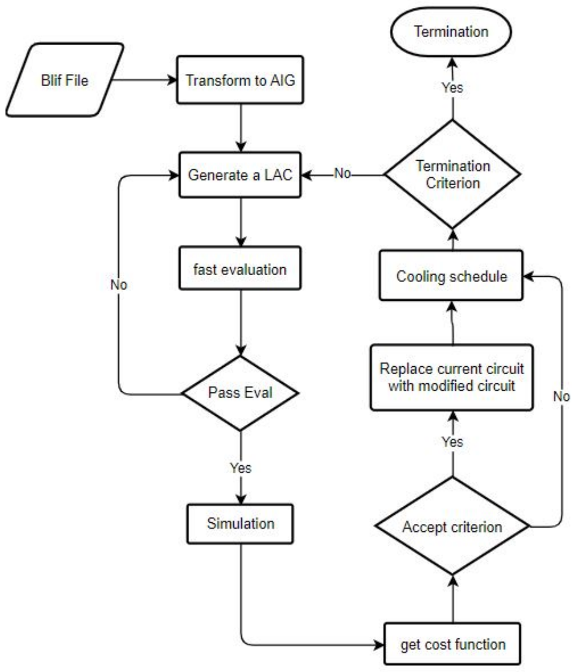


Figure 1: Overall ALS Flow

to generate relatively good candidate LACs in each iteration. Then, we use a multi-objective simulated annealing (MOSA) core to decide whether this LAC would be applied to perturb the current circuit.

2.2. Exact vs. Simulation-based Error Evalutaion

The runtime of precise computation of error caused by LACs grows exponentially with input size. [4] introduced the idea of computing the local erroneous input pattern of a logic node by a miter, and deriving the number of satisfying assignments in the fan-in cone. The error rate of this gate's output is obviously an overestimate of the real error rate at POs due to possible reconvergence in the fan-out cone and the observability don't cares (ODCs) of this node in the Boolean network.

We experimented with this method by knowledge compilation (KC) but found some implementation issues. First, KC would be intractable if the size of the cone is too large. Second, the ignorance of ODCs is temporary since the network structure is dynamically changing. Third, the computation of multiple LACs would be difficult due to the fact that the error rates of each node could affect each other.

Therefore, we decided to estimate the error rate by performing random simulations on the entire circuit, in accordance with previous works. The empirical mean would be an unbiased estimator of the real error rate, which converges to the true value if the variance decreases at a comparable speed. However, in practice, performing a large number of simulations will eventually slow down the program, especially for large circuits. In order to resolve this problem, we take advantage of the following two strategies:

2.2.1. Parallel Simulation. Parallel simulation is crucial for reducing runtime. In our program, we decide to utilize the additional space created in the AIG nodes (`iTemp` in each `Abc_Obj_t`) to store the simulation results. This method not only speeds up our program but also prevents us from creating any additional memory space.

2.2.2. Early Stop. In our program, we perform two kinds of early stops to lower the run time of the simulation progress. If the current circuit's error is larger than the tolerable threshold, we will terminate our simulation progress and abandon this LAC. Another type of early stop is to terminate the simulation progress if the error rate has converged into a range that is small enough.

2.3. Multi-objective Simulated Annealing

2.3.1. Cost Function Design. ALS could be viewed as a kind of constraint optimization, while we aim to minimize the area under the given error tolerance constraint. Simulated Annealing (SA) is a stochastic approach that aims to minimize the given objective function, with has non-zero probability dependent on the cost gain for accepting uphill (worse) configurations. Constraint handling is crucial in MOSA's optimization. A simple method is to directly reject solutions with violations; however, it may lead to discontinuous state space that may lead the optimization process to get stuck at some point. We adopt the soft penalty function in the error rate dimension by smoothing the Rectified Linear Units (ReLU) and setting the breakpoint at the given error threshold. Intuitively, it is expected that the area reduction ratio shall be larger at lower error rates (e.g., 0% to 1%) compared to higher error rates (e.g., 4% to 5%). Thus, the area dimension is characterized by a concave function, such that the contour line could describe the desired behavior, as shown in Figure 2.

2.3.2. Neighborhood structure. The neighborhood structures we selected can be described by $N = E \cup A$. Each is chosen with equal probabilities. *Exact Optimization* Neighborhoods E collects a set of exact logic minimization which can be performed on AIG, such as AIG orchestration [6], balancing, sweeping and fraiging. *Approximate* Neighborhoods A includes:

- *Constant propagation*. Randomly choose a node and replace one of its fan-ins with a constant one or zero.
- *Input Edge Flip*. Randomly choose a node and randomly complement one of its fan-in edges.
- *Functionally Equivalent Candidates (FECs)*. First, we perform a number of simulations proportional to the circuit size and classify AIG nodes into different equivalent classes using their simulation behaviors. Then, in every class, we replace other nodes with the node that is relatively positioned at the forefront in

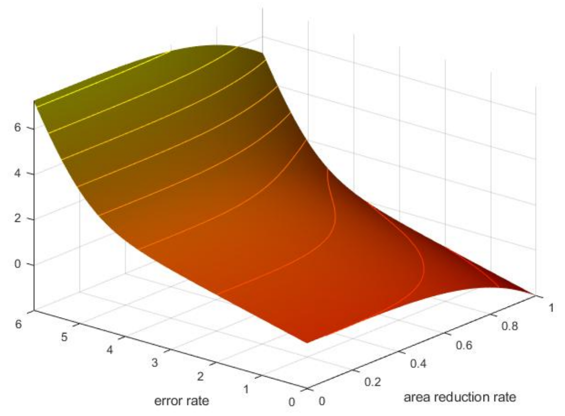


Figure 2: Cost surface design with an error rate threshold at 5%.

the DFS list within this class. This type of change won't increase the error rate by a lot since most of the gates in the same class are almost the same. However, it lowers the area tremendously, saving up over half of the area in some of the experiments. We innovated this idea based on the previous courses on FRAIG.

- *Add Node with PI*. Randomly choose a node and create a node (AND, OR, or XOR) between it and one of its fan-outs. Another fan-in of the created node is one of the PI. This LAC can generate more complex patterns and have the chance to decrease the error rate and make MOSA achieve a better trade-off between the error rate and the area.

2.3.3. Choosing "Better" Nodes by Fast Simulation. In our initial experiments, if we apply the LACs for a randomly selected node, we usually get a large error rate, so the LACs would be discarded. In this way, we seldom perform our LACs and the solution and area couldn't be minimized. Therefore, we innovated a strategy in which we make a *few* times of node selections. For each choice, we do a fast evaluation by simulating the error a few times. We accept the node if the error rate is tolerable and continue the MOSA procedure, in which the error rate will be evaluated more accurately. After this improvement, the engine usually chooses a "good" node to do the LAC and shows a better area-error rate tradeoff.

3. Experimental Results

3.1. Experimental Setup

We implement our proposed algorithm in C++ as an extension of the synthesis tool ABC [7]. All tests were run on 13th Gen Intel(R) Core (TM) i9-13900K with 128GiB memory. ER is chosen as our error metric and 1 % is chosen as our error bound in the experiments. As for the area estimation, we use the technology mapping library *mcnc.genlib* and *mcnc-aig.genlib*, and use the command `map` in ABC. The area ratio (the area of the ALS circuit over the original one) is used to evaluate the quality of the ALS result. The EPFL Combinational Benchmark Suite [8] is used for our experiment's circuits. These settings align with the setting in [1]. Since SA is a randomized algorithm, we report the average and the best results for running 3 rounds. We compare our results on both random control and arithmetic circuits.

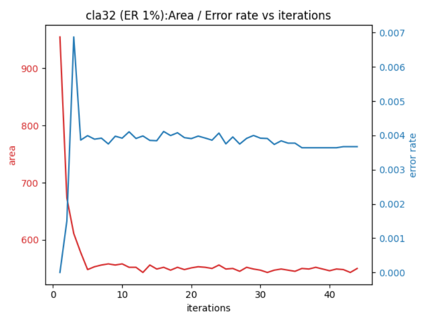
Here are some important parameters for our experiment. The criteria for early stop is that the rate fluctuates less than 0.001% for 300 consecutive cycles. The number of simulations used for classifying FECs is 10% of the circuit size. For the annealing schedule, we cool down the temperature with a decreasing rate from 0.85 to 0.95 and back to 0.85. This is similar to the TimberWolf cooling schedule, which can be considered a successful scheme reported in terms of balancing final quality and runtime.

TABLE 1: Area ratio after ALS on EPFL benchmarks. Min and avg are the statistics of our results.

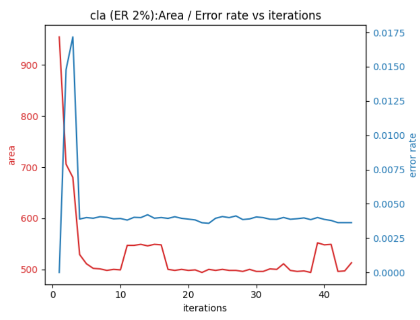
Category	Circuit	Area	ALSRAC	min	avg
Arithmetic	alu4	2770	70.46%	63.14%	64.73%
	c1908	919	75.24%	77.26%	79.04%
	c2670	1306	66.75%	65.08%	66.49%
	c3540	1881	92.89%	89.00%	90.16%
	c5315	2717	87.91%	85.90%	87.03%
	c7552	2894	80.30%	80.13%	80.79%
	c880	640	90.48%	78.91%	82.97%
	cal32	955	59.72%	50.99%	52.91%
	ksa32	1703	70.17%	30.18%	30.46%
	mtp8	1319	95.31%	81.96%	82.97%
	rca32	666	91.31%	71.92%	74.17%
	wal8	1157	80.80%	92.65%	92.91%
Random Control	arbiter	46910	53.06%	3.56%	3.57%
	cavlc	2665	93.07%	76.89%	78.12%
	i2c ctrl	4760	78.41%	57.18%	61.88%
	int2float	892	89.29%	61.88%	63.12%
	mem ctrl	177620	70.56%	23.07%	25.40%
	priority	3459	9.09%	1.39%	1.46%
	router	973	57.69%	1.03%	1.03%
	voter	54126	97.77%	58.57%	59.53%

3.2. Discussion

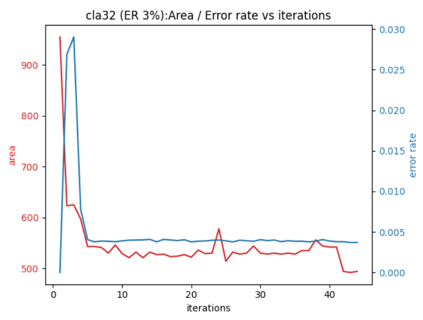
3.2.1. Performance. The result of our proposed method is shown in Table 1, with the result in [1]. For arithmetic circuits, our best case quality is better than ALSRAC's quality for all the cases except `c1908` and `wal8`. This phenomenon doesn't change a lot when it comes to average cases, only with `c7552` losing by 0.39%. For random control circuits, our performance outperforms ALSRAC dominantly in every case. For `arbiter`, `priority`, and `router`, we saved more than 95% of the original area, which is pretty astonishing to us.



(a) 1% ER threshold.



(b) 2% ER threshold.



(c) 3% ER threshold.

Figure 3: ALS area and error rate vs. Iterations with different ER threshold on `cla32` Benchmark.

To find out what happened, we dug into the circuit's structure and found that many of the outputs are very similar to constant 1 or constant 0. Therefore, the error rate won't increase a lot if we simply change an output into constant. Interestingly, we found out that this result can be presented even without our constant propagation. This is because adding nodes randomly would reduce the logic to constant too. The cost function of our MOSA engine would tend to make this decision due to the significant area reduction.

3.2.2. Effect of interleaving E and A. Another interesting observation is that in many cases, performing exact optimization can only reduce the area to some extent. However, if we apply a few LACs, the exact optimization will continue to lower the area since there is additional design space to explore. This observation is practically effective for the benchmark `arbiter`, as none of the gates will be reduced by using only exact optimization. However, the area ratio can be reduced to 10% if our LACs are interleaved within exact optimizations.

3.2.3. Effect of the cost function design. The cost function is designed to prevent the error rate from exceeding the threshold, and the effect can be seen in Figure 3. As MOSA optimizes both area and error rate, once the current circuit has reached the maximum tolerable error, the optimization direction will attempt to maintain the low area while minimizing the error rate. The final error rate is almost half of the given threshold, which indicates that our design approach has way more space for optimization.

In some specific cases, we found that the ALS result would be similar, independent of the given ER threshold. We focus on `cla32` since its error rate fluctuates at 0.4%. This phenomenon still happens when we increase the threshold to 2% and 3%, as shown in Figure 3b and 3c. In these cases, the error rate in the first few iterations indeed climbs almost up to the threshold error rate but decreases to near 0.4% at the end. We suggest that the design state space exhibits a good local minimum at around 0.4% error rate and thus the result is easy to reproduce. In conclusion, the cost function may be extremely sensitive to the underlying circuit.

4. Conclusion, Work Division

In this report, we proposed a method for ALS by using MOSA engine to apply various LACs, which shows better performance over the existing ALS tool ALSRAC. The authors would like to thank Prof. Jie-Hong Roland Jiang for offering such a nice comprehensive course for logic synthesis and verification. The GitHub of our method is available at this GitHub link.
B09901047, Survey (Boolean Rewriting), LACs
B09901156, Survey (High-level ALS), MOSA
B09901189, Survey (Netlist Transformation), Simulation

References

[1] C. Meng, W. Qian, and A. Mishchenko, "Alsrac: Approximate logic synthesis by resubstitution with approximate care set," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2020.

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

[3] G. Ammes, W. L. Neto, P. Butzen, P.-E. Gaillardon, and R. P. Ribas, "A two-level approximate logic synthesis combining cube insertion and removal," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 41, p. 5126–5130, nov 2022.

[4] Y. Wu and W. Qian, "An efficient method for multi-level approximate logic synthesis under error rate constraint," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2016.

[5] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "Salsa: Systematic logic synthesis of approximate circuits," in *DAC Design Automation Conference 2012*, pp. 796–801, 2012.

[6] Y. Li, M. Liu, M. Ren, A. Mishchenko, and C. Yu, "Dag-aware synthesis orchestration," *arXiv preprint arXiv:2310.07846*, 2023.

[7] R. K. Brayton and A. Mishchenko, *ABC: an Academic Industrial-Strength Verification Tool.* 2010.

[8] L. G. Amarù, P.-E. Gaillardon, and G. D. Micheli, "The epfl combinational benchmark suite," 2015.