Statistically Certified Approximate Logic Synthesis

Gai Liu and Zhiru Zhang
School of Electrical and Computer Engineering, Cornell University, Ithaca, NY
{gl387, zhiruz}@cornell.edu

Abstract

Approximate logic synthesis generates inexact implementations of logic functions in exchange for better design qualities such as area, timing and power consumption. However, the error behavior of the approximate circuits (e.g., error rate or error magnitude) depends heavily on the specific synthesis technique as well as the input vectors, hindering end users from confidently adopting approximate designs. In this paper, we propose a statistically certified approximate logic synthesis framework using techniques from stochastic optimization, and integrate it into a state-of-the-art parallelized technology mapper. During the synthesis process, our framework continuously monitors the quality of the generated designs using statistical testing, leading to approximate designs that adhere to user-specified error constraints with a high confidence level. Experimental results demonstrate up to 10x area and timing improvements over the exact counterparts with an average of 0.2% deviation from the exact outputs.

1. Introduction

Approximate computing is an emerging design paradigm aiming to improve the quality of result (QoR) such as area, timing and power consumption at the cost of carefully-controlled errors [6]. Representative application domains include data mining and machine learning, where infrequent errors at the outputs do not significantly degrade user experience, and image/signal processing, where errors of small magnitude are not perceivable by the end users [4, 13].

This work focuses on approximate logic synthesis, a key step to automating the process of creating approximate circuits based on an exact logic function. Existing approaches of approximate logic synthesis usually simplifies a logic network in a technology-independent manner by removing logic gates or connections between gates, subject to constraints on error rate and/or error magnitude [9, 10, 12].

We use Figure 1 as an example to illustrate a typical approximate logic synthesis flow and its drawbacks. Given an initial design as shown in Figure 1 with an error rate constraint of 10%, one possible simplification that an existing synthesis algorithm would make is to remove gate g_2. This move decreases both the gate count and output level of the logic network by one, but generates incorrect results for two input patterns $a = 0/1, b = 0, c = 1, d = 1$ (out of the 16 input combinations in total). The synthesis engine then samples N number of input vectors uniformly and simulates the circuit with these input vectors to estimate the error rate, where N is small relative to the total number of possible input combinations. If none of the sampled input vectors leads to the two erroneous outputs[1], the synthesis engine would incorrectly conclude that the generated design satisfies the 10% error rate constraint.

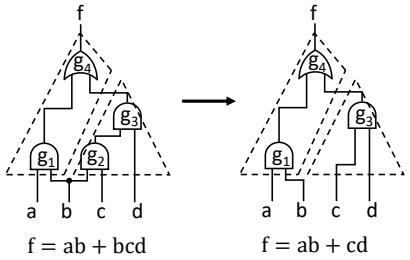

Figure 1. An example illustrating the drawback of existing approximate synthesis techniques using sample statistic approach — The approximate synthesis algorithm removes gate g_2, reducing the gate count and output level both by one. However, this approximation does not reduce the area or depth of the final netlist after mapping to 3-input LUTs (as highlighted with dashes).

In addition to inaccurate characterization of the error behavior, the current approach may also fail to improve the actual quality of results of the circuit after technology mapping. With the example in Figure 1, neither the area nor the timing is improved after the approximation. We attribute such unfavorable outcome to three major drawbacks of the conventional approximate logic synthesis techniques:

Disconnect from downstream flow While existing approaches are effective in simplifying the gate-level logic network, they usually do not take into account the impact of logic simplification on the QoR after mapping to a specific technology library, such as lookup tables (LUTs) for FPGAs.

Misrepresentation of realistic input distributions Realistic datasets rarely follow uniform distribution. Using test vectors drawn from uniform distribution can lead to incorrect conclusions on error metrics. In addition, synthesis techniques that explicitly rely on this assumption of input distribution will not work for other types of input distributions.

Lack of statistical rigorousness Using random samples to measure the error metrics of an approximate design is inherently a statistical process. In the language of statistical inference, existing methods of equating sampled error behavior with the true error behavior fail to distinguish the difference between sample and population statistics. For example, although the sample mean of a design's error magnitude correlates with its population mean, they are in general not equal to each other due to statistical noise. Capturing such noise

[1] Under uniform sampling, the likelihood of observing such an event is $(14/16)^N$, which is significant for small values of N.

through the lens of statistical testing is crucial in assuring a high-confidence evaluation of the error metrics.

To overcome these obstacles, we propose a statistically certified approximate logic synthesis (SCALS) framework, which extends PIMap [8], a state-of-the-art logic synthesis and mapping framework to generate approximate designs. Following the techniques in PIMap, SCALS couples logic simplification with technology mapping to iteratively simplify the circuit. During the synthesis process, SCALS continuously monitors the quality of the intermediate design points using the technique of statistical testing, leading to a final approximate circuit that adheres to user-specified error constraints.

Our primary technical contributions are as follows:

- We are the first to apply statistical testing techniques to approximate logic synthesis to generate approximate designs with user-specified statistical guarantees.
- We propose a generic approximate logic synthesis framework that can effectively handle various input distributions, error metrics and technology targets.
- We show that our approach achieves better QoR than existing synthesis techniques for both ASIC and FPGA targets while providing statistical guarantees on the error metrics.

The rest of the paper is organized as follows: Section 2 reviews the related work; Section 3 describes the key techniques of our approach; Section 4 presents the experimental results, followed by conclusions in Section 5.

2. Related Work

We first summarize the existing research directions on approximate logic synthesis, followed by describing a logic synthesis and technology mapping framework that our work builds on.

2.1 Prior Arts on Approximate Logic Synthesis

We review a subset of representative work on approximate logic synthesis. One line of research uses formal methods to enforce that the outputs of an approximate circuit satisfy the error requirement for all input patterns. Venkataramani, et al. [15] propose SALSA, which is an approximate logic synthesis framework based on approximation don't cares to simplify circuits using traditional don't care based optimization techniques. Miao, et al. [9] formulate approximate two-level logic synthesis under error magnitude constraint as the Boolean relation minimization problem, and devise an efficient heuristic algorithm for iteratively refining the magnitude-constrained solution to arrive at a solution also satisfying the error rate constraint. This work is extended by [10] to handle multi-level logic networks. Chandrasekharan et al. [3] present an automatic synthesis approach using and-inverter graphs (AIGs) based rewriting. The proposed method selects cuts in the AIG that are on the critical paths and replace them with constant zeros. The error behavior of the synthesized circuit is verified using satisfiability solver. Compared to a sampling-based approach, the aforementioned approaches enforce strong restrictions on the simplifications that can be made on the circuit, thus limiting the area and timing improvement of the generated circuit. In addition, these approaches often incur significant runtime overhead due to extensive use of satisfiability solvers.

Another line of research evaluates the error behavior of an approximate circuit using random input samples. Shin, et al. [12] propose techniques for synthesizing approximate two-level circuits by selectively complementing the output values of the minterms to reduce the number of literals in the SOP representation. Venkataramani, et al. [14] describe approximation techniques by identifying signal pairs in the circuit that assume the same value with high probability, and substitute one for the other. Wu, et al. [17] propose techniques for approximate logic synthesis under error rate constraint, which iteratively removes literals from the logic expression of selected nodes in a Boolean network. They further extend their techniques for mapping to FPGAs by removing wires in the LUT network and changing the functionality of LUTs [18].

2.2 Logic Synthesis and Technology Mapping Framework for Synthesizing Exact Designs

Our work builds on a recently proposed cross-stage logic synthesis framework named PIMap [8]. It is a logic synthesis and technology mapping framework for generating exact designs targeting FPGAs, which couples logic transformations and technology mapping under an iterative improvement framework to minimize the circuit area for LUT-based FPGAs. The core of PIMap is an iterative area minimization framework that repeats three major steps: (1) proposing logic transformation moves, (2) evaluating the quality of the move through technology mapping, and (3) determining whether to accept the proposed move. Such an iterative cross-stage synthesis framework is useful for addressing the three major drawbacks of the conventional approximate logic synthesis techniques discussed in Section 1. PIMap makes use of a collection of logic transformation moves such as balancing and logic rewriting. At each iteration, PIMap randomly selects one logic transformation, and applies it to the logic network. It then maps the transformed logic network to LUTs, measures the quality of proposed transformation, and uses the Markov Chain Monte Carlo method [5] to probabilistically determine whether to accept the proposed move. To reduce runtime overhead, PIMap automatically extracts a number of non-overlapping sub-netlists from a mapped netlist, and optimize them in parallel through multithreading.

3. SCALS Techniques

We first describe the approximate logic synthesis problem. We then introduce our proposed techniques on generating approximate logic and hypothesis testing of the error metrics.

3.1 Problem Formulation

We study the problem of synthesizing approximate designs with user-specified error constraints under a given probability distribution for the input values. Specifically, given a combinational logic network composed of technology-independent

logic gates[2], SCALS minimizes the area and/or delay of the generated design after mapping to a specific technology library for either LUT-based FPGAs or ASIC standard cells.

For error constraints, we focus on two representative error metrics, error rate and mean relative error magnitude, although our framework is generic enough to handle other types of error metrics. Error rate (ER) is defined as the probability that the approximate circuit generates incorrect outputs when the input test vectors are drawn randomly from a specific input distribution. Mean relative error magnitude (MREM) measures the population mean of the error magnitude relative to the exact output[3]. SCALS makes use of the input distribution for measuring the error metrics as well as guiding the synthesis process. The input distribution can be specified in two different ways. The user can supply a known probability distribution for the input vectors based on profiling results of the realistic datasets. Alternatively, when the input distribution is not known a priori, SCALS directly draw samples from the design-specific input vectors to measure the error metrics. Together with the error constraint, the user also provides SCALS with a statistical confidence level to specify the required level of statistical significance during hypothesis testing [16].

3.2 Overall Flow

SCALS extends the PIMap flow (Section 2.2) to approximate logic synthesis. Figure 2 shows the overall flow of SCALS, where the modifications to PIMap and additional steps in SCALS are highlighted. Starting with an initial gate-level logic network, we first map it to the targeted technology. The mapped netlist is then partitioned into a set of sub-netlists, each of which contains a predefined number of standard cell components (when mapping to standard cell library) or LUTs (when mapping to FPGA technology). These extracted sub-netlists are then independently optimized in parallel using the iterative logic optimization routine (Section 3.3).

The optimized sub-netlists are recombined and evaluated using statistical hypothesis testing (Section 3.4). If the new netlist satisfies the error requirement, SCALS accepts and uses it for the next trial. Otherwise, SCALS discards the current design point and proceeds to the next trial using the output from the previous trial. We note that SCALS is unlikely to get stuck at an infeasible design point for consecutive trials due to two sources of randomness. Firstly, the sub-netlist extraction algorithm uses random seeds to generate different sets of sub-netlists in different trials, uncovering various network structures for optimization. Secondly, the iterative logic optimization routine randomizes the selection of transformation moves as well as the logic gate to be simplified, which effectively explores a wider range of potential simplifications than a fixed sequence of logic transformations.

[2] Our approach can be generalized to handle sequential circuit by optimizing the combinational blocks between register boundaries individually.

[3] The maximum error magnitude (MEM) is another commonly used error metric. Approximate logic synthesis under an MEM constraint can be formulated as a Boolean relation problem and solved using SAT solvers [9]. While our framework can integrate SAT solvers to handle MEM constraints, we focus our experiments on statistical error metrics in this work.

3.3 Iterative Logic Optimization

SCALS uses a collection of logic transformation moves denoted as the set $T = E \cup A$. For each logic transformation move i in T, we associate it with a probability p_i. During each iteration of the iterative logic optimization step, we select one logic transformation i from T with probability p_i.

Exact transforms E is the set of logic transformations that do not alter the functionality of the input design. The set of exact logic transformations contain three commonly used moves, i.e., $E = \{$`balance`, `rewrite`, `refactor`$\}$. These transformations either balance the logic depths of difference paths in the logic network, or reduce the gate count in the network by logic rewriting [11].

Approximate transforms A is the set of logic transformations that simplify the logic network but may generate incorrect outputs. In SCALS, this set includes three types of moves, i.e., $A = \{$`reduce`, `flip`, `add`$\}$. Figure 3 illustrates the effects of the three approximate logic transformations. The `reduce` transformation randomly selects one logic gate in the logic network and removes a randomly-selected fanin of the logic gate. If the selected logic gate has only one fanin before removal, then the logic gate itself will be removed, with its fanin node directly connected to the fanouts of the original logic gate. Similar to `reduce`, the `flip` transformation randomly selects one logic gate in the logic network and inverts one of its randomly-selected fanins. Finally, the `add` transformation adds a two-input logic gate with randomly-selected functionality to the logic work, where its two fanin nodes and one fanout node are randomly selected from the existing nodes in the logic network.

We choose to use a stochastic approach to simplify the logic circuit instead of fixed heuristics mainly for two reasons. First, a stochastic approach provides a unified framework towards logic simplification that can handle various input vectors, error metrics and technology targets. Second, random perturbations of the logic network can explore various promising design points over a large number of iterations.

Evaluating and accepting a transformation After applying the selected logic transformation to the logic network, SCALS immediately maps the logic network to the targeted technology and measures the area, denoted as $Area_m$. If an approximate logic transformation is applied, SCALS also estimates the impact of the approximation on the primary outputs through logic simulation. Note that SCALS does not perform hypothesis testing per iteration to reduce runtime overhead. Instead, SCALS conducts hypothesis testing after each trial.

Since the iterative logic optimization routine operates on extracted sub-netlists, it is important to be able to estimate error behaviors at the primary outputs based on the local outputs of the sub-netlists. We use the following procedure to achieve this goal. Using input vectors drawn from the user-specified input distribution, SCALS first simulates the entire design to obtain a set of test vectors for the nodes serving as the inputs to the sub-netlists. These test vectors will then be used to simulate all sub-netlists in parallel, and generate lo-

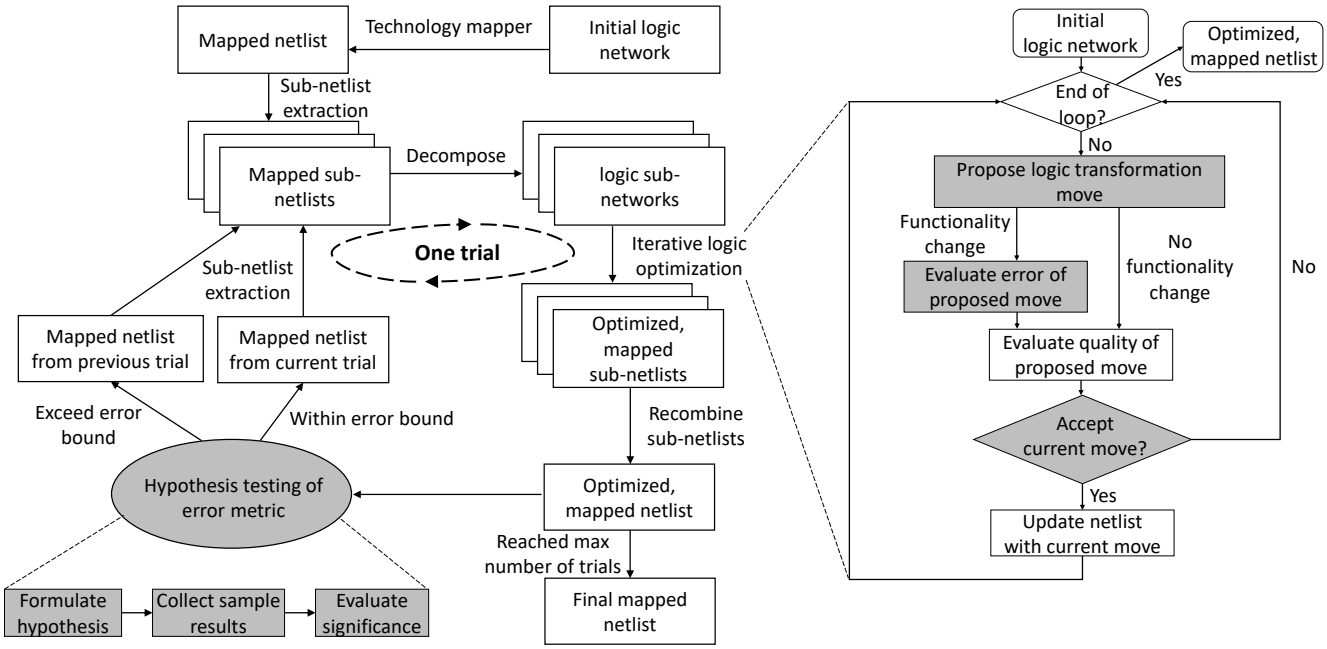

Figure 2. Overall flow of SCALS.

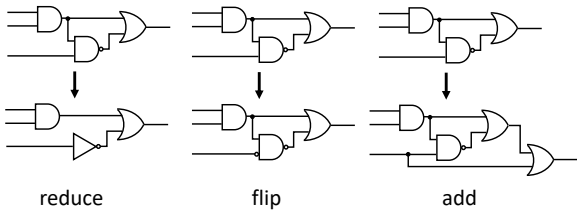

Figure 3. Illustration of approximate transformations.

cal outputs for each sub-netlist. Afterwards, we go through the transitive fanins of each PO O_i. If any of these transitive fanins is an output of an extracted sub-netlist S_i, and S_i generates inexact result during simulation of the sub-netlist, then we conservatively infer that O_i is inexact. We then use this information to calculate the error metrics (EM) such as ER and MREM at the primary outputs. We note that this is a conservative estimation for O_i due to the possible scenario where the error at S_i is not observable at O_i given the specific test pattern. Nonetheless, this error estimation scheme provides a quick way of inferring global error behavior from sub-netlists without the need of simulating the entire design every iteration.

After obtaining the post-mapping area $Area_m$ and the error metric EM, SCALS calculates a quality metric of the current transformation as a weighted sum of $Area_m$ and EM, i.e., $Q_{curr} = \alpha \cdot Area_m + \beta \cdot EM$. Q_{curr} is then compared with the quality metric from the previous iteration (i.e., Q_{prev}). Specifically, we use the Markov Chain Monte Carlo (MCMC) method to probabilistically determine whether to accept the proposed move [5]. In particular, we employ the Metropolis-Hastings algorithm for calculating the acceptance probability [7]. The Metropolis-Hastings algorithm dictates that if the quality of the current move is better than the previous one, we accept the current move unconditionally. Otherwise, we accept the move with a probability of $e^{-\gamma(Q_{curr}/Q_{prev})}$, which decreases exponentially as Q_{curr} increases.

3.4 Hypothesis Testing of Error Constraint

At the end of each trial, SCALS evaluates the error metric of the generated approximate design using statistical hypothesis testing [16]. Given an error metric EM and the constraint $EM \leq C$, we formulate the null hypothesis $H_0 : EM > C$ and the alternative hypothesis $H_1 : EM \leq C$. To show that the error metric stays within the constraint, the null hypothesis needs to be rejected under a user-specified confidence level CL. After selecting an appropriate test statistic for the error metric, SCALS generates N number of samples by simulating the approximate design using input vectors drawn from the corresponding input distribution. Using the test statistic and the observed samples, we evaluate the probability (P-value) of observing the output samples assuming the null hypothesis holds true. Finally, SCALS makes conclusion on the null/alternative hypotheses based on the test outcome.

Testing error rate constraint SCALS uses the binomial test [16] to examine the error rate of an approximate design. Given a hypothesis that an approximate design has an error rate of p under certain input distribution, SCALS first samples N outputs from the circuit using independently-drawn input vectors and compare them with the expected correct outputs. We denote the observed number of incorrect outputs as n. If the null hypothesis is true, then the number of incorrect outputs in the N output samples should follow the binomial distribution, i.e., $n \sim B(N, p)$. Using the binomial test, SCALS evaluates the P-value of the observed event, and determines the outcome of the test by comparing the P-value and CL.

For the scenario in Figure 1 with $N = 4$ and $n = 0$, the P-value of this event (observing four consecutive correct outputs given an error rate of 10%) is $0.9^4 = 65.6\%$, which

is above the significance level $1 - CL = 5\%$. Consequently, the test fails to reject the null hypothesis. That is, there is no sufficient evidence to conclude that the error rate of the approximate design is below 10%.

Testing mean relative error magnitude constraint In statistical inference, a T-test [16] is used for testing the population mean with unknown variance. Given a total of N samples with a sample mean $\bar{X}$, a sample standard deviation S and the population mean μ to be tested, the test statistic $T = \frac{\bar{X}-\mu}{S/\sqrt{N}}$ follows the T-distribution [16], i.e., $T(t) \sim \frac{\Gamma(\frac{v+1}{2})}{\sqrt{v\pi}\Gamma(\frac{v}{2})}(1 + \frac{t^2}{v})^{-\frac{v+1}{2}}$, where Γ is the gamma function, and v is the number of degrees of freedom ($v = N - 1$). SCALS uses the T-test for testing a mean relative error magnitude constraint. Similar to testing the error rate, SCALS first samples the relative error magnitude using N input vectors drawn from the user-supplied input distribution. SCALS then calculates the P-value using the test statistic and the observed samples. Based on the calculated P-value and CL, SCALS either accepts or rejects the null hypothesis.

Extension to other error metrics While SCALS focuses on error rate constraint and mean relative error magnitude constraint, we can extend SCALS to handle other types of error constraint by drawing the connection between the specific error metric and its corresponding test statistic. For example, another interesting test is on whether a given approximate design generates unbiased outputs under certain input distribution. Unbiased designs are particularly useful for applications where an approximate module is used repeatedly because the errors could potentially cancel each other out. Such a test can be formulated as testing whether the population mean of the error magnitude is equal to zero, which can be carried out using the T-test as detailed above. In scenarios where we are interested in constraining the variance of the error, we can use χ^2 test [16] to examine whether an error metric satisfies a constraint on the population variance.

4. Experimental Results

We implement the SCALS techniques in C as extensions to the ABC logic synthesis framework [2]. We target mapping to both ASIC standard cell library and FPGA LUTs using a combination of the EPFL combinational benchmark suite [1] and the MCNC benchmark suite [19]. For ASIC targets, we map to the MCNC generic standard cell library [19]. For FPGAs, we target 4-input or 6-input LUTs. Of course, our approach also supports mapping to other standard cell libraries and LUT architectures. We report the synthesis results under four representative input distributions including uniform, Gaussian, exponential, and bimodal distributions. For each design, SCALS runs for 20 trials, and each trial contains 100 iterations of the iterative logic optimization routine. We choose $\alpha = 0.05$ and $\beta = 0.95$ when calculating the quality metric as detailed in Section 3.3. We assign equal probabilities for selecting each of the six transformation moves in T as discussed in Section 3. Each hypothesis testing step uses a sample size of 10000 test vectors to determine the validity of the hypotheses. We partition the original design to up to 16 sub-netlists, where each sub-netlist contains up to 100 LUTs. We run our experiments on eight machines, each with a quad-core Xeon CPU operating at 2.7GHz.

We demonstrate the effectiveness of SCALS from three aspects. We first show that SCALS significantly improves the area and delay of the designs over their exact counterparts, where we compare against the best-known 6-LUT mapping results from the EPFL benchmark suite. Secondly, we show that when compared to the representative state-of-the-art techniques [17, 18], our approach achieves better QoR under the same error constraints for both ASIC and FPGA targets, while providing additional statistical guarantees that the existing approaches do not offer. Finally, we present the case study of an approximate FIR filter that achieves significant QoR improvements under error magnitude constraint.

4.1 Arithmetic Circuit under Relative Error Magnitude Constraint

Table 1 shows the area and depth comparisons with the exact counterparts for the arithmetic benchmarks in the EPFL benchmark suite [1] under mean relative error magnitude constraint for various input distributions. The baseline designs are the best-known 6-LUT mapping results according to the EPFL records [1]. As an example to demonstrate the effectiveness of SCALS, we enforce a mean relative error magnitude constraint of $\frac{1}{2^9}$ ($\approx 0.2\%$). That is, the magnitude of the output from the approximate designs cannot exceed 0.2% of the correct output value on average. SCALS also works well with other values of the error magnitude constraint. All designs pass hypothesis testing on error magnitude given a confidence level of 0.95.

We observe that some designs (e.g., `hyp`, `sqrt`, and `square`) are highly error tolerate, requiring only 10% to 15% of the area of the original designs to meet the error constraint. We also observe significant depth reduction for the majority of the designs due to the simplified logic structure in the approximate designs. SCALS achieves similar QoRs regardless of the different input distributions, showing that SCALS flexibly adapts to the input characteristics and generate high-quality designs for various types of input distributions.

4.2 Random-Control Circuit under Error Rate Constraint

Table 2 shows the area and depth reduction of the EPFL random-control designs targeting 6-input LUTs under various input distributions. In this experiment, we require that the error rates of the generated designs do not exceed 1% with a confidence level of 0.95. The baseline designs are again the best-known mapping results from the EPFL records [1]. Similar to the results of arithmetic designs, the QoR improvement of the approximate designs is design specific.

Noticeably, for design `priority`, a 128-to-7 priority encoder, SCALS achieves almost 10x reduction in both area and depth. This is because SCALS is able to generate a highly efficient design by exploiting the logic structure of a priority encoder. Specifically, since the higher order inputs are prioritized over the lower order inputs, the errors related to the lower order inputs will often be masked and become un-

Table 1. Area and depth reduction for arithmetic circuit under mean relative error magnitude constraint with various input distributions on EPFL benchmarks — `Exact` = Exact designs from the EPFL benchmark record; `SCALS` = Approximate designs synthesized using our method; `Size` = Area of the circuit measured as the number of 6-input LUTs; `Dpt` = Depth of the circuit in terms of 6-input LUTs; `Ratio` = The ratio of size of depth of `SCALS` over `Exact`. The average error magnitude is constrained to be within $\frac{1}{2^9}$ ($\approx 0.2\%$) of the correct output value. Numbers in bracket indicate the size/depth ratio over the corresponding exact design. All designs pass hypothesis testing with a confidence level of 0.95.

	Exact		SCALS							
			Uniform		Gaussian		Exponential		Bimodal	
Designs	Size	Dpt	Size	Dpt	Size	Dpt	Size	Dpt	Size	Dpt
adder	192	64	137 (0.71)	10 (0.16)	139 (0.72)	12 (0.19)	129 (0.67)	14 (0.22)	142 (0.74)	9 (0.14)
shifter	512	4	461 (0.90)	4 (1.00)	478 (0.93)	4 (1.00)	487 (0.95)	4 (1.00)	452 (0.88)	4 (1.00)
divisor	3268	1208	3232 (0.99)	1068 (0.88)	3122 (0.96)	842 (0.70)	1580 (0.48)	268 (0.22)	2651 (0.81)	699 (0.58)
hyp	40406	4532	3662 (0.09)	142 (0.03)	4201 (0.10)	152 (0.03)	4115 (0.10)	134 (0.03)	4028 (0.10)	133 (0.03)
log2	6574	119	6401 (0.97)	108 (0.91)	6529 (0.99)	118 (0.99)	6564 (1.00)	118 (0.99)	6485 (0.99)	117 (0.98)
max	523	189	184 (0.35)	19 (0.10)	180 (0.34)	16 (0.08)	130 (0.25)	3 (0.02)	158 (0.30)	22 (0.12)
mult	4923	90	1337 (0.27)	36 (0.40)	1959 (0.40)	45 (0.50)	1043 (0.21)	35 (0.39)	1057 (0.21)	25 (0.28)
sine	1229	55	1219 (0.99)	54 (0.98)	1218 (0.99)	54 (0.98)	1196 (0.97)	54 (0.98)	1205 (0.98)	55 (1.00)
sqrt	3077	1106	338 (0.11)	112 (0.10)	236 (0.08)	77 (0.07)	344 (0.11)	114 (0.10)	352 (0.11)	108 (0.10)
square	3246	74	490 (0.15)	19 (0.26)	867 (0.27)	27 (0.36)	771 (0.24)	20 (0.27)	2909 (0.90)	39 (0.53)

Table 2. Area and depth reduction for random-control circuit under error rate constraint with various input distributions on EPFL benchmarks— `Exact` = Exact designs from the EPFL benchmark record; `SCALS` = Approximate designs synthesized using our method; `Size` = Area of the circuit measured as the number of 6-input LUTs; `Dpt` = Depth of the circuit in terms of 6-input LUTs; `Ratio` = The ratio of size of depth of `SCALS` over `Exact`. The error rate is constrained to be within 1%. Numbers in bracket indicate the size/depth ratio over the corresponding exact design. All designs pass hypothesis testing with a confidence level of 0.95.

	Exact		SCALS							
			Uniform		Gaussian		Exponential		Bimodal	
Designs	Size	Dpt	Size	Dpt	Size	Dpt	Size	Dpt	Size	Dpt
arbiter	409	23	251 (0.61)	13 (0.57)	170 (0.42)	7 (0.30)	159 (0.39)	6 (0.26)	153 (0.37)	5 (0.22)
alu ctrl	27	2	27 (1.00)	2 (1.00)	26 (0.96)	2 (1.00)	26 (0.96)	2 (1.00)	26 (0.96)	2 (1.00)
cavlc	101	6	100 (0.99)	5 (0.83)	99 (0.98)	6 (1.00)	100 (0.99)	5 (0.83)	99 (0.98)	6 (1.00)
decoder	270	2	270 (1.00)	2 (1.00)	270 (1.00)	2 (1.00)	270 (1.00)	2 (1.00)	270 (1.00)	2 (1.00)
i2c controller	227	7	205 (0.90)	6 (0.86)	209 (0.92)	7 (1.00)	168 (0.74)	4 (0.57)	168 (0.74)	4 (0.57)
Int2float	28	6	26 (0.93)	6 (1.00)	25 (0.89)	6 (1.00)	26 (0.93)	4 (0.67)	23 (0.82)	4 (0.67)
mem ctrl	2354	22	2086 (0.89)	15 (0.68)	1895 (0.81)	14 (0.64)	1316 (0.56)	11 (0.50)	1468 (0.62)	8 (0.36)
priority	110	26	12 (0.11)	3 (0.12)	13 (0.12)	2 (0.08)	11 (0.10)	3 (0.12)	54 (0.49)	21 (0.81)
router	52	6	31 (0.60)	2 (0.33)	31 (0.60)	2 (0.33)	35 (0.67)	4 (0.67)	32 (0.62)	2 (0.33)
voter	1301	17	1299 (1.00)	17 (1.00)	1298 (1.00)	17 (1.00)	1299 (1.00)	17 (1.00)	1299 (1.00)	17 (1.00)

observable at the output. Consequently, SCALS utilizes this logic structure to largely simplify the logic for the lower order inputs without incurring significant errors at the output. On the other hand, designs such as `alu ctrl` that have simple internal logic, SCALS can not simplify its logic at all. The original design of `alu ctrl` only needs 27 LUTs to implement the 26-output function, meaning that almost every LUT directly generates one output. Consequently, any simplifications of the logic network will likely lead to errors at the primary outputs, leaving little room for approximation.

4.3 Comparison with Existing Work Targeting ASICs

We compare our approach with the single-selection algorithm in [17], which represents the state-of-the-art approximate logic synthesis method for ASICs. Since the single-selection algorithm focuses on area reduction after technology mapping, we mainly compare the post-mapping area results of the generated approximate designs. In this experiment, we use seven MCNC benchmarks that are also used in [17].[4]. We set the error rate constraint to 1% in Table 3. The baseline designs in Table 3 are the initial exact designs generated from ABC [2].

Compared to the single-selection algorithm, our approach achieves higher area reduction across all benchmarks, with an average improvement of 37% over the baseline designs, while [17] achieves an average improvement of 17%. When directly comparing with the single-selection algorithm, our approach reduces the area by 24% on average across the seven benchmarks considered here. We observe similar area improvements for other values of error rates. For example, at 5% error rate, the designs generated from our approach are 10% smaller on average than the designs from [17]. While the approach in [17] does not report the delay numbers of the generated designs, we observe that our approach achieves smaller delay than the baseline designs, mainly due to the simplified logic structure in the approximate designs. The runtime of our approach is in general on the same order of the approach in [17], and the average runtime across the seven benchmarks is smaller than that of the approach in [17].

4.4 Comparison with Existing Work Targeting FPGAs

Table 4 shows the area reduction when compared with a state-of-the-art FPGA approximate logic synthesis algorithm [18]

[4] The single-selection algorithm [17] reports area results for 12 benchmarks in total. However, we are only able to obtain seven of the benchmarks since the other five benchmarks are not publicly available.

Table 3. Comparison with a state-of-the-art approximate logic synthesis method for ASIC targeting area minimization [17] on MCNC benchmarks — `Base` = Initial exact designs reported by [17]; `[17]` = Results of the single-selection algorithm in [17]; `SCALS` = Results from our approach; `Time` = Runtime in seconds. Both `[17]` and `SCALS` enforce a 1% error rate constraint. The designs generated by SCALS pass the hypothesis testing at a confidence level of 0.95. The area and delay numbers are normalized to the area and delay of a unit size inverter, respectively.

	Base		Wu, et al. [17]			SCALS				SCALS vs. [17]
Designs	Delay	Area	Time (s)	Area	Ratio vs. Base	Time (s)	Delay	Area	Ratio vs. Base	Ratio vs. [17]
c880	40.4	599	93	497	0.83	507	36.7	494	0.82	0.99
c1908	60.6	1013	394	654	0.65	93	45.2	324	0.32	0.50
c2670	67.3	1434	702	935	0.65	137	35.4	748	0.52	0.80
c3540	84.5	1615	172	1554	0.96	101	54.0	1396	0.86	0.90
c5315	75.3	2432	263	2352	0.97	251	47.5	2245	0.92	0.95
c7552	159.8	2759	533	2527	0.92	476	155.7	2396	0.87	0.95
alu4	51.5	2740	1000	2433	0.89	607	45.3	1099	0.40	0.45
geomean					0.83				0.63	0.76

Table 4. Comparison with a state-of-the-art approximate logic synthesis method for FPGA targeting depth-constrained area minimization [18] on MCNC benchmarks — `Base` = Exact designs generated from ABC [2]; `Size` = Area of the circuit measured as the number of 4-input LUTs; `Dpt` = Depth of the circuit in terms of 4-input LUTs. Both approaches generate designs with an error rate constraint of 5%. The designs generated by SCALS pass the hypothesis testing at a confidence level of 0.95. Both approaches do not increase the depth of the baseline designs. No runtime information is provided for these set of designs in [18].

	Base		Wu, et al. [18]		SCALS			SCALS vs. [18]
Designs	Depth	Size	Size	Ratio vs. Base	Depth	Size	Ratio vs. Base	Ratio vs. [18]
c432	10	97	79	0.81	10	55	0.57	0.70
c880	8	128	102	0.80	8	107	0.84	1.05
c1908	9	122	50	0.41	9	88	0.72	1.76
c2670	7	295	242	0.82	7	224	0.76	0.93
c3540	12	346	325	0.94	11	305	0.88	0.94
c5315	9	503	468	0.93	9	439	0.87	0.94
c7552	8	593	486	0.82	8	440	0.74	0.91
alu4	7	710	483	0.68	7	411	0.58	0.85
alu2	12	160	136	0.85	11	135	0.84	0.99
apex6	6	253	197	0.78	6	210	0.83	1.07
dalu	11	425	349	0.82	11	329	0.77	0.94
geomean				0.77			0.76	0.98

over a set of benchmarks from the MCNC benchmark suite. The baseline designs are the exact designs generated from ABC [2]. Following the requirement in [18], both approaches enforce a 5% error rate constraint and require that the generated approximate designs do not increase the depth of the baseline designs. We measure the size of the designs as the number of 4-LUTs. SCALS generates smaller designs when compared with [18] for eight out of the 11 designs, showing the effectiveness of SCALS when compared with the existing synthesis technique.

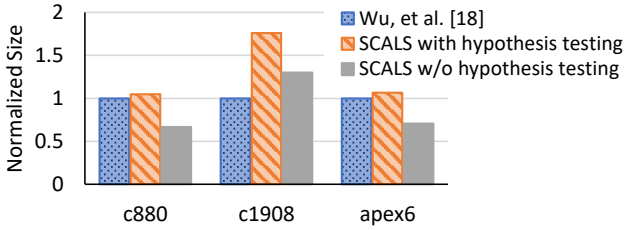

Figure 4. Area comparison after disabling hypothesis testing.

We note that although SCALS and [18] both enforce a 5% error rate constraint, it is still not an apple-to-apple comparison since the requirement for an approximate design to pass the test of SCALS is stricter than that in [18]. The hypothesis testing step in SCALS would potentially reject an approximate design that passes a simple sampling-based test as in [18]. To understand the impact of applying hypothesis testing on the design QoR, we look into the three designs in Table 4 that SCALS fails to improve over [18]. Figure 4 shows the size of the generated designs from SCALS with and without the hypothesis testing step. The size of the designs are normalized to the corresponding design generated by [18]. We observe that disabling the hypothesis testing step indeed leads to smaller designs.

4.5 QoR vs. Confidence Level Tradeoff

Table 5. Synthesis results under different confidence levels — `Size` = Area of the circuit measured as the number of 6-input LUTs; `Dpt` = Depth of the circuit in terms of 6-input LUTs; `CL` = Confidence level for error rate during hypothesis testing. The designs are generated under 1% error rate constraint after 10 trials.

	CL = 0.90		CL = 0.95		CL = 0.99	
Designs	Size	Dpt	Size	Dpt	Size	Dpt
arbiter	348	17	352	17	354	21
mem ctrl	2309	21	2315	21	2354	22
priority	14	2	14	4	16	4
router	32	3	33	3	34	3

Table 5 shows the impact of confidence level during hypothesis testing on the area and depth of the final designs using four random-control circuits under error rate constraint.

A higher confidence level requires stronger evidence for certifying that an error constraint is honored. Consequently, a higher confidence level will lead to more conservative designs as shown in Table 5, which provides a tradeoff between QoR improvement and statistical confidence of the error behavior.

4.6 Design Study: Approximate FIR Filter

We provide a design study using approximate multipliers and adders generated using SCALS from design-specific input vectors to construct a three-tap finite response (FIR) filter. The FIR filter has a passband between 100 to 200Hz and a stopband from 300 to 500Hz. To supply the input patterns for SCALS, we use realistic input waveform that contains two major frequency components at 100Hz and 300Hz with Gaussian noise sampled at 1KHz. All inputs to the multipliers and adders are 64-bit fixed-point numbers.

In Table 6, the area and delay numbers are the post-synthesis results normalized to the area of a unit-size inverter. The measured MREM is verified at a confidence level of 0.95. The generated approximate FIR filter is 5x smaller and 2x faster than the exact counterpart, while maintaining similarly small MREM (0.012%) of its basic building blocks.

Table 6. Area and delay comparison between an exact FIR filter and an approximate FIR filter from SCALS — `Exact` = Exact designs generated using ABC's optimization script `resyn2` and technology mapper `map`; `SCALS` = Approximate designs synthesized using our method; `Area` = ASIC circuit area; `Delay` = ASIC circuit delay; `MREM` = Measured mean relative error magnitude. The area and delay numbers are normalized to those of a unit size inverter.

	Multiplier		Adder		FIR	
	Exact	SCALS	Exact	SCALS	Exact	SCALS
Area	55544	12058	2476	465	171234	33607
Area ratio	1.00	0.22	1.00	0.19	1.00	0.20
Delay	215.2	104.6	204.6	32.7	226.9	112.5
Delay ratio	1.00	0.49	1.00	0.16	1.00	0.50
MREM		0.009%		0.001%		0.012%

5. Conclusions

We propose SCALS, a statistically certified approximate logic synthesis framework based on parallelized stochastic optimization. SCALS effectively handles various error metrics, technology targets and input distributions in a unified framework, and provides statistical guarantee that the generated designs adhere to user-specified error constraints. SCALS generates approximate designs that are up to 10x smaller and faster than their highly optimized exact counterparts. It also achieves better design QoR across different technology targets when compared to existing approximate logic synthesis techniques.

6. Acknowledgements

This work was supported in part by NSF Award #1618275, a DARPA Young Faculty Award D15AP00096, and a research gift from Xilinx, Inc. We thank the anonymous reviewers for their helpful feedback.

References

[1] L. Amarú, P.-E. Gaillardon, and G. De Micheli. The EPFL Combinational Benchmark Suite. *International Workshop on Logic & Synthesis (IWLS)*, 2015.

[2] Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification, Release 60413. `http://www.eecs.berkeley.edu/~alanmi/abc/`.

[3] A. Chandrasekharan, M. Soeken, D. Große, and R. Drechsler. Approximation-aware Rewriting of AIGs for Error Tolerant Applications. *Int'l Conf. on Computer-Aided Design (ICCAD)*, 2016.

[4] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. Analysis and Characterization of Inherent Application Resilience for Approximate Computing. *Design Automation Conf. (DAC)*, 2013.

[5] C. J. Geyer. Practical Markov Chain Monte Carlo. *Statistical Science*, pages 473–483, 1992.

[6] J. Han and M. Orshansky. Approximate Computing: An Emerging Paradigm for Energy-Efficient Design. *2013 18th IEEE European Test Symposium (ETS)*, pages 1–6, 2013.

[7] W. K. Hastings. Monte Carlo Sampling Methods using Markov Chains and Their Applications. *Biometrika*, 57(1):97–109, 1970.

[8] G. Liu and Z. Zhang. A Parallelized Iterative Improvement Approach to Area Optimization for LUT-Based Technology Mapping. *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2017.

[9] J. Miao, A. Gerstlauer, and M. Orshansky. Approximate Logic Synthesis under General Error Magnitude and Frequency Constraints. *Int'l Conf. on Computer-Aided Design (ICCAD)*, 2013.

[10] J. Miao, A. Gerstlauer, and M. Orshansky. Multi-Level Approximate Logic Synthesis under General Error Constraints. *Int'l Conf. on Computer-Aided Design (ICCAD)*, 2014.

[11] A. Mishchenko, S. Chatterjee, and R. Brayton. DAG-Aware AIG Rewriting a Fresh Look at Combinational Logic Synthesis. *Design Automation Conf. (DAC)*, 2006.

[12] D. Shin and S. K. Gupta. Approximate Logic Synthesis for Error Tolerant Applications. *Design, Automation, and Test in Europe (DATE)*, 2010.

[13] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan. AxNN: Energy-Efficient Neuromorphic Systems using Approximate Computing. *Int'l Symp. on Low Power Electronics and Design (ISLPED)*, 2014.

[14] S. Venkataramani, K. Roy, and A. Raghunathan. Substitute-and-Simplify: A Unified Design Paradigm for Approximate and Quality Configurable Circuits. *Design, Automation, and Test in Europe (DATE)*, 2013.

[15] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. SALSA: Systematic Logic Synthesis of Approximate Circuits. *Design Automation Conf. (DAC)*, 2012.

[16] B. J. Winer, D. R. Brown, and K. M. Michels. *Statistical Principles in Experimental Design*. McGraw-Hill, 1971.

[17] Y. Wu and W. Qian. An Efficient Method for Multi-Level Approximate Logic Synthesis under Error Rate Constraint. *Design Automation Conf. (DAC)*, 2016.

[18] Y. Wu, C. Shen, Y. Jia, and W. Qian. Approximate Logic Synthesis for FPGA by Wire Removal and Local Function Change. *Asia and South Pacific Design Automation Conf. (ASP-DAC)*, 2017.

[19] S. Yang. *Logic Synthesis and Optimization Benchmarks*. Microelectronics Center of North Carolina (MCNC), 1991.