CIRCA: Towards a Modular and Extensible Framework for Approximate Circuit Generation

Linus Witschen, Tobias Wiersema, Hassan Ghasemzadeh Mohammadi, Muhammad Awais, Marco Platzner
Paderborn University, Germany
Email: {witschen, tobias.wiersema, hgm, mawais, platzner}@mail.upb.de

Abstract—**Existing approaches and tools for the generation of approximate circuits often lack generality and are restricted to certain circuit types, approximation techniques, and quality assurance methods. Moreover, only few tools are publicly available. This hinders the development and evaluation of new techniques for approximating circuits and their comparison to previous approaches. In this paper, we first analyze and classify related approaches and then present CIRCA, our flexible framework for search-based approximate circuit generation. CIRCA is developed with a focus on modularity and extensibility. We present the architecture of CIRCA with its clear separation into stages and functional blocks, report on the current prototype, and show initial experiments.**

Keywords-**Approximate Computing, Framework, Pareto Front, Accuracy**

I. Introduction

Approximate computing subsumes a variety of approaches that trade-off computational accuracy for improvements in hardware area, delay, and/or energy consumption. This trade-off becomes possible since for many applications approximate results are indeed good enough and often hard to distinguish from accurate results. Examples for such applications can be found in the processing of audio, image, and video data as well as in data mining and machine learning. While computer scientists and engineers have been dealing with limited accuracy and suboptimal quality of results for a long time, e.g., due to the limited precision of data types or in developing heuristics, recent research has been focusing on more aggressive approximation techniques at all levels of the computing stack, from programming languages down to semiconductor technology.

Our interest is in methods for the automated approximation of combinational and sequential circuits and accelerators, respectively. Related approaches presented in literature employ iterative design processes, typically search-based. A search space is spanned by iteratively creating different versions of approximate circuits, selecting one of these versions by heuristics, and checking whether the version still satisfies user-provided error bounds. The difficulty in studying and comparing related approaches is that they are designed, or at least described, as monolithic blocks with interwoven phases for search, approximation, and quality checking. Furthermore, only few frameworks are openly available for experimentation, which hinders the development of new techniques for approximating circuits.

In this paper we present CIRCA, our framework for approximate circuit generation. CIRCA is developed to be modular and extensible and will be made publicly available. We argue that such a framework is of great value for the approximate computing community, since it facilitates the rapid implementation and evaluation of new ideas and also enables fair experimental comparisons of alternative approximation techniques.

The remainder of this paper is structured as follows: In Section II we discuss the related work. In Section III we provide a classification of related work according to several categories, state requirements for a flexible approximation framework, and, finally, present the architecture of our own CIRCA framework. Since CIRCA is work in progress, we give a short overview over its current state in Section IV and also show some experiments conducted with the prototype. Section V concludes the paper and describes future work.

II. Related Work

In the following, we provide a brief overview over related approaches for the automated generation of approximate circuits. We focus on the approximation techniques and how they are iteratively applied to generate one or more approximated circuit variants.

Venkataramani et al. proposed SASIMI [1] which uses a substitute-and-simplify approximation technique. Near-identical signal pairs are identified, i.e., two signals which show a similar behavior, and one is substituted with the other. Subsequently, the required logic can be simplified. SASIMI evaluates and ranks signal pairs with a heuristic function including area and delay parameters. With a gradient ascent technique, more accurately hill climbing, the highest ranked pair is selected for substitution. The process is iterated until the user-defined quality constraints are violated. For the resulting circuit, power and area are reported. The quality of the approximate circuits is determined by testing. Additionally, the authors suggest the concept of *quality configurable circuits*, which are circuits that can operate in either an accurate or approximate version.

Venkataramani et al. also presented SALSA [2] which forms a so-called *quality constraint circuit* by providing the original and the approximated circuit, which initially is identical to the original circuit, with the same input and feeding the outputs of the circuits into a quality function that checks whether the given error bound holds. Forcing the error bound to hold, SALSA works backwards and applies standard don't care logic optimization techniques to reduce the area of the approximated circuit. Thus, the technique creates approximated combinational circuits that adhere to the error bound by construction.

Similar to SALSA, the approach of Chandrasekharan et al. [3] employs a setup with a quality constraint circuit but formally verifies the error constraint by combinational equivalence checking using a SAT solver. The approach represents a circuit's logic function as AND-Inverter graph (AIG) and employs AIG re-writing as approximation technique, i.e., setting nodes to constant zero. Among all possible cuts on the critical paths of the circuit, the one with smallest cut size is selected for re-writing. This heuristic is greedily iterated until there is no more possibility for re-writing without violating the error bound. For the resulting circuit, area and delay estimates gained by ABC [4] are reported.

Nepal et al. [5] proposed a methodology called ABACUS which transforms a circuit into an Abstract Synthesis Tree (AST). In an iterative approach, AST transformations are applied to create approximated circuit candidates. The accuracy of the candidates is evaluated by testing, area and power characteristics via ASIC synthesis using a standard cell library. The resulting three metrics are then combined into a fitness function. The candidate with the best fitness is greedily selected as next current circuit. This heuristic process runs for a user-defined number of iterations. Eventually, a Pareto front of designs is given, trading off accuracy for power.

The SCALS framework presented by Liu and Zhang [6] maps an initial gate-level logic network to a target technology. In an iterative process, sub-netlists are extracted from the mapped netlist and are subjected to randomly chosen approximations or optimizations. The candidates are then evaluated by a function including the error, gained through a testing approach, and the area. A Metropolis-Hastings algorithm steers the candidate selection and search until a predefined number of iterations is reached. Additionally, the user can specify a confidence interval for the estimated error.

The ASLAN framework [7] by Ranjan et al. is, to the best of our knowledge, the only presented framework able to approximate sequential circuits while guaranteeing error bounds. In a first step, ASLAN extracts combinational subcircuits amenable to approximation. Then, a search space is generated by creating approximated versions for the subcircuits that vary in their local error constraints and estimated energy consumption. The applied approximation techniques are precision scaling and SALSA [2], although the authors also mention the applicability of other techniques. Finally, ASLAN employs hill climbing to find a locally optimal combination of approximated subcircuits. In each iteration, subcircuit versions with larger error bounds are considered and the combination resulting in the greatest energy savings is selected if the circuit adheres to the global error bound. Otherwise, the next-best combination of subcircuits is picked. Quality assurance relies on a so-called *sequential quality constraint circuit (SQCC)* that raises a flag in case the error bound is violated. Since ASLAN deals with sequential circuits, time frame expansion is used to unroll both the original and the approximated circuit until they finish their computations. The resulting Boolean expression is then formally verified with a SAT solver.

III. Towards a Framework for Aproximate Circuit Generation

A. Classification of Existing Frameworks

Table I presents our attempt to classify the presented frameworks. The challenges are two-fold, first to identify meaningful and orthogonal categories and, second, to retrieve the required information from related works. Table I comprises categories in four groups: the input, the circuit generation/synthesis step, the output, and whether the framework has been made publicly available.

Regarding the input, the first category is the circuit type. Most of the frameworks approximate combinational circuits, while ASLAN was developed for sequential circuits. It has to be noted that it is not necessarily the characteristic of the input circuit that determines its type in our classification. Rather, an approximation technique for sequential circuits means that at least the approximation technique or the quality assurance step (as for ASLAN) are considering the clocked nature of the circuit. For example, several frameworks use sequential circuits such as FIR filters as benchmarks, but restrict the approximation to the datapath and do not report on testing the resulting circuit for a sequence of clock cycles or formally verifying it. Hence we classify these approaches as "combinational" in Table I. For ABACUS we are somewhat uncertain how to classify it, since the mentioned approximation techniques are clearly for sequential behavior, e.g., loop transformations, but a corresponding quality assurance was not detailed.

As input model, SASIMI, SALSA, and the approach of [3] rely on gate level netlists. SCALS takes technology-mapped netlists as input, ASLAN begins with circuits described in structural HDL, and ABACUS, operating on a more abstract level, requires behavioral HDL or RTL code. Another issue is how the user can control the error. Most frameworks allow for specifying an error bound, often in several error metrics such as error rate or average error. SALSA and ASLAN define qualitiy functions and quality evaluation circuits, respectively, which encode error bounds. Again, ABACUS is different as it generates a Pareto front showing reasonable trade-offs between accuracy and power. The circuit generation process is controlled by a user-specified

Table I
OVERVIEW OVER PRESENTED FRAMEWORKS FOR APPROXIMATE CIRCUIT GENERATION.

Category	SASIMI [1]	SALSA [2]	AIG re-writing [3]	ABACUS [5]	SCALS [6]	ASLAN [7]
Circuit Type	Combinational	Combinational	Combinational	Combinational + sequential (?)	Combinational	Sequential
Input model	Gate netlist	Gate netlist	Gate netlist/AIG	Behavioral HDL	Gate/LUT netlist	Structural HDL + annotations
Error control	Error bound	Quality function	Error bound	# Iterations	Error bound	Quality evaluation circuit
Search method	Heuristic (hill climbing)	–	Heuristic (greedy)	Heuristic (greedy)	Heuristic (Metropolis-Hastings)	Heuristic (hill climbing)
AC technique	Substitute-and-simplify	Approx. don't care	AIG re-writing	AST transforms	Logic transforms	Precision scaling
Quality assurance	Testing	By construction	Formal verification	Testing	Testing	Formal verification
Output	Approx. circuit	Approx. circuit	Approx. circuit	Pareto front	Approx. circuit	Approx. circuit
Output model	Gate netlist	Gate netlist	Gate netlist (AIG)	Behavioral HDL	Gate/LUT netlist	Structural HDL
Target technology	Standard cell	Standard cell	Techn. independent	Standard cell	Std. cell/LUT-based	Standard cell
Publicly available	–	–	Yes	Yes	–	–

number of iterations. Generally, more iterations lead to more approximations and, in turn, can add non-dominated designs to the Pareto front with larger errors.

The second group of rows in Table I characterizes the circuit generation step, split into the three categories search, approximation technique, and quality assurance. All techniques rely on heuristics for search, the exception being SALSA that does not apply a search technique but systematically iterates over the outputs of the approximated circuit to apply don't care optimization. Consequently, SALSA creates circuits that adhere to the error bound by construction, making a subsequent quality assurance step obsolete. ASLAN and the framework in [3] formally verify circuit quality, which is time-consuming but provides a much stronger statement about quality than the testing approaches used in SASIMI, ABACUS, and SCALS. It has to be noted that there are error metrics, e.g., the average-case error [8], for which formal verification cannot be used.

The next group of categories characterizes the result produced by the frameworks. Mostly, the tools return one approximated circuit in form of a gate level netlist or structural HDL. ABACUS, however, returns a set of designs in behavioral HDL that form a Pareto front with respect to accuracy and power. Since the results of all frameworks are either netlists or synthesizable hardware descriptions they can potentially target standard cell and FPGA technology. In contrast, with the category "target technology" we refer to the technology used to get estimates for area, delay, and power during the synthesis process. Here, most frameworks target standard cell libraries with the exception of [3], where the AIG representation and ABC functions, respectively, are employed to retrieve technology-independent estimates for area and delay.

Finally, only the authors of ABACUS [5] and the work described in [3] decided to make their frameworks publicly available.

B. Requirements for a Flexible Framework

Our analysis of related frameworks and the attempt to categorize them showed that all these approaches have been developed for specific circuit types and approximation techniques. In particular, circuit generation is typically described as a monolithic block with interwoven phases for approximation, search, and assuring quality. Moreover, only few frameworks are openly available for experimentation. This situation severely hampers the development and evaluation of new techniques for approximating circuits, and the comparison to existing ones.

With our work, we aim at overcoming these shortcomings and provide a flexible framework for approximate circuit generation. As a starting point for this development, we take our classification of Table I. This classification provides several categories and shows that many of these are largely orthogonal, giving rise to a reasonable structuring of our framework. Generally, we envision a framework that is:

- *General:* The framework should not be restricted to certain circuit types, error metrics, approximation and search techniques, or specific target technologies.
- *Modular:* The framework architecture should enable the exchange of certain processing steps without affecting other steps.
- *Compatible:* The framework, in particular its input and outputs, should connect to other, widely-used academic and commercial front-end and back-end tools.
- *Extensible:* The framework should facilitate the swift implementation and evaluation of new techniques.
- *Open source:* The framework should be publicly available and allow other researchers to use, modify, and extend it.

C. The CIRCA Framework

We are developing the CIRCA approximation framework with an architecture shown in Figure 1. We target search-

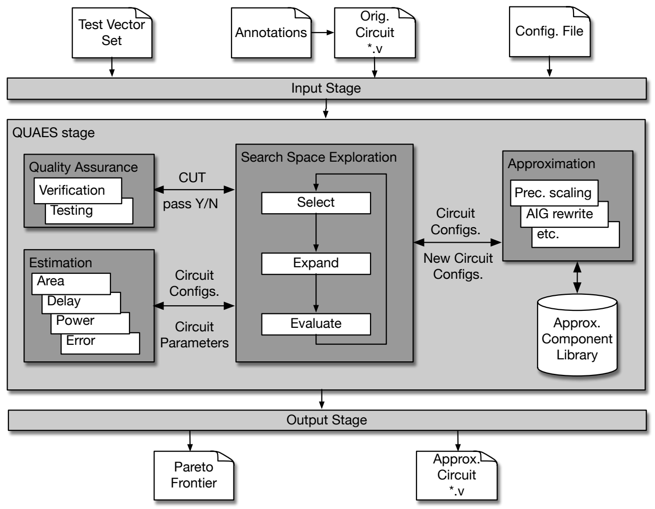

Figure 1. Architecture of the CIRCA approximation framework.

based approximation approaches with different approximation techniques and both formal verification and testing for circuit validation. The framework is divided into tree stages: the input stage, the QUAES stage, and the output stage. The input stage processes user-provided information, which in any case comprises an annotated Verilog description of the original, exact circuit and a configuration file. The configuration file defines the employed functionality in the QUAES stage and specifies the target metric as well as the quality constraints for the approximated circuits. The annotations in the Verilog code identify subcircuits amenable to approximation. Since our framework supports also testing as quality assurance technique, the input can include a test vector set.

The main stage for generating approximate circuits is the QUAES stage. The split of the corresponding functionality into the four blocks Quality Assurance, Approximation, Estimation, and Search Space Exploration (QUAES) is driven by our classification of related frameworks in Subsection III-A and is key to achieving a modular and extensible architecture. We denote the subcircuits annotated in the input Verilog code as *candidates*, and the different approximated versions of these candidates as *variants*. The overall circuit with instantiated variants for the candidates is called a *circuit configuration*.

The structure of the search space exploration is kept rather general, and thus, sufficiently flexible to support a wide range of search algorithms. The search space is iteratively explored by creating and visiting configurations, which form the nodes of the search space, by applying the three steps Select, Expand, and Evaluate. The evaluation step takes a set of configurations and determines estimated values for parameters of interest. Typically these parameters cover metrics related to area, delay, and power but can also include estimates of the error metrics. The actual estimation functions are encapsulated in the estimation block to clearly separate estimation from the search space exploration. The select step receives a set of configurations, each with an annotated vector of estimated parameters, and selects the next configuration to be further considered, i.e., to be expanded. This selection relies on a search heuristic. The selected configuration is subsequently validated. To this end the configuration, now called Circuit-under-Test (CUT), is sent to the quality assurance block. This block either employs formal verification or testing to decide if the CUT satisfies the quality constraints and passes the check or not. If a CUT fails quality assurance, the select step will, depending on the configuration file, either abort the search or pick the next best configuration for validation. In the latter case, the search terminates if there are no more valid configurations.

If a CUT passes validation, the configuration and thus the search space is expanded by creating a number of new configurations. For creating new configurations the approximation block applies certain approximation techniques. Depending on the actual configuration of the framework, one or more candidates of the configuration can be approximated and one or more approximation techniques can be applied. The approximation block accesses a library of approximated subcircuits, which is beneficial for two reasons: First, it is rather likely that one circuit component will be approximated for several times. This happens when the overall circuit contains identical candidates, e.g., multiple occurrences of an 8-bit unsigned adder. Storing the approximated versions of such components and retrieving them the next time can greatly save computations. Second, there are already libraries available of approximated components [9], [10] which can then be leveraged by CIRCA.

During the QUAES stage, all validated configurations are stored. The output stage performs post-processing on these configurations to return either the best approximated circuit, e.g., a circuit that respects the error constraints and has minimal area or energy, or a Pareto-filtered set of circuits.

IV. CIRCA Implementation

The implementation of CIRCA is work in progress. The framework is mainly being coded in Python and will be made open-source. In this section, we first present the current state of CIRCA including the implemented techniques for the different phases of the QUAES stage and then show some experiments that demonstrate CIRCA's current functionality.

A. Current State

Our implementation follows the architecture presented in Subsection III-C and currently supports the approximation of sequential circuits using hill climbing search, precision scaling (PS) and approximation-aware AIG re-writing [3] as approximation techniques, formal verification based quality assurance, and the worst-case error as well as the bit-flip error [3] as error metrics.

The search process generates for each candidate a new variant with a slightly degraded quality, i.e., increased error bound. For each variant, individual error bounds are provided by the search step which the new variant has to adhere to. This ensures that the search space is expanded in a controlled manner. Then the variants are evaluated using a heuristic function taking an area estimate into account, based on ABC's technology mapper for 4-LUT FPGA logic blocks. The variant with the biggest area reduction is selected and installed in the currently selected configuration which is validated subsequently. If no more variants are left whose instantiation satisfy the error bound and lead to an area reduction, the search terminates.

The current implementation of CIRCA differs from ASLAN [7], which is the only related framework for approximating sequential circuits, in two important aspects. First, in the search phase we implement a hill climbing version that expands the search space step-by-step. In contrast, ASLAN generates an entire search space before starting the actual search procedure. Our approach of selectively expanding the search space shows two advantages: We can reduce the computational effort since only the variants of interest are being generated, and we do not have to prematurely bound the search space.

The second difference is that besides time frame expansion as used by ASLAN we can additionally employ inductive verification to assure the quality. Time frame expansion unrolls both the correct and the approximated sequential circuit until they reach the end of their computations. Inductive verification is stronger as it provides general guarantees, not just for a finite time frame. The use of inductive verification enables us to validate a larger set of sequential circuits, including run-to-completion circuits that raise a done signal, streaming circuits that produce results at regular intervals, and circuits for which we have to check the error constraint in every single clock cycle.

B. Experiments

We have performed a number of experiments with our current CIRCA prototype as described in Subsection IV-A. The target metric is hardware area expressed in the number of used FPGA 4-LUTs as reported by ABC [4].

Table II
SEQUENTIAL BENCHMARK CIRCUITS

Circuit Name	Description	#4-LUTs*	#Candidates
butterfly$^{\text{II}}$	Operation used in FFT	7221	8
fir_gen$^{\text{II}}$	FIR filter 4-tap	5438	7
fir_pipe_16†	FIR filter 16-tap	8768	23
rgb2ycbcr$^{\wr}$	Color-space transformation	4527	5
ternary_sum_nine$^{\text{II}}$	Adder tree	1483	4
weight_calculator	Industrial scale	1872	4

* ABC estimation after mapping. $^{\text{II}}$ From Meyer-Baese [11].
† From VTR [12]. $^{\wr}$ OpenCores JPEG Encoder [13].

Table II lists the benchmarks we have used for experimentation. The benchmarks are sequential circuits with areas ranging from 1483 to 8768 FPGA 4-LUTs. Adders and multipliers in the data path have been manually annotated as candidates. We have varied the error bound from 0.25% to 2.0%, expressed in percentage of the circuit's maximum possible output value, and employed ABC's *dprove* command for circuit verification. For each benchmark, we have run the approximation flow ten times and determined the median as representative result.

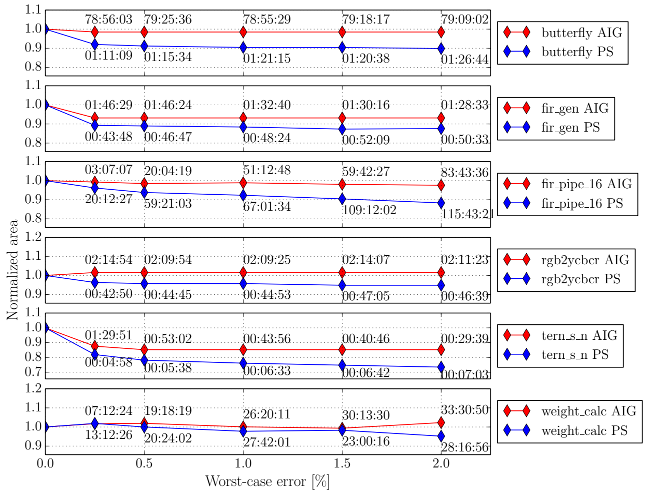

Figure 2. Normalized area in dependence of the specified error bound. The labels show the average runtimes of the approximation process in the format hours:minutes:seconds.

Figure 2 shows the resulting area, normalized to the area of the original circuit, over the worst-case error bound. We could achieve area savings for each benchmark by either applying precision scaling or AIG re-writing while guaranteeing the user-specified error bounds. The maximum saving of $\approx$ 26% has been achieved for *ternary_sum_nine PS*. Comparing the two approximation techniques leads to the conclusion that, in general, precision scaling achieves higher area savings than AIG re-writing.

As can be seen in Figure 2, the area is not strictly decreasing and even an increase is possible, e.g., for the *weight_calculator* benchmark with a worst-case error bound of 0.25%. This is caused by our prototype's assumption that a local improvement in the target metric, i.e., area for one candidate, will result in a globally improved target metric, i.e., area for the overall circuit. However, for hardware area, this is not compulsory since the mapping tools can exploit different optimization opportunities in different circuits. The approximated *weight_calculator* circuit at 0.25% error is obviously Pareto-dominated in the area-error space and thus not a reasonable design choice. This issue could easily be addressed by accepting further approximated circuits only if they are not dominated in the area-error space.

The label at each data point in Figure 2 represents the average runtime of the particular benchmark. The quality assurance block has been identified as the dominating part for the runtime and ranges from a couple of minutes to several days, depending on the complexity of the verification problem. Somewhat against intuition, the results reveal that more relaxed error bounds do not necessarily lead to longer runtimes, e.g., *fir_gen AIG*. This is caused by the randomness involved in the search which influences the path taken through the search space which, in turn, determines the number of verifications, and thus, influences the runtime.

V. Conclusion and Future Work

In this paper, we have presented the CIRCA framework for approximate circuit generation. CIRCA is developed to be a modular and extensible framework and will be made publicly available. We have elaborated on the architecture of CIRCA, which was driven by analyzing the commonalities and differences of related frameworks, and then reported on the current state of the prototype implementation as well as on initial experiments.

We plan future work along several lines: First, we will continue with the implementation of alternative techniques in the QUAES stage, i.e., for approximation, search, quality assurance, and estimation. Besides covering also delay and energy as target metrics we are particularly interested in the trade-offs between more accurate estimations and the required computational effort. Second, in the output stage we plan to connect to back-end synthesis tools, such as FPGA vendor tools or the Synopsys Design Compiler to be able to get accurate circuit characteristics and actually run the approximate circuits, at least in FPGA technology. Third, we will look into approaches to automatically identify subcircuits amenable to approximation in the input specification. This would relieve the user from the tedious process of marking potential candidates and make such a framework wider applicable. Finally, we aim at collecting a sufficiently large suite of benchmark circuits to be bundled with the framework.

Acknowledgment

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901). Calculations leading to the results presented here were performed on resources provided by the Paderborn Center for Parallel Computing.

References

[1] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits," in *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '13, pp. 1367–1372, IEEE, 2013.

[2] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: Systematic logic synthesis of approximate circuits," in *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pp. 796–801, ACM, 2012.

[3] A. Chandrasekharan, M. Soeken, D. Groesse, and R. Drechsler, "Approximation-aware rewriting of aigs for error tolerant applications," in *Proceedings of the 35th International Conference on Computer-Aided Design*, ICCAD '16, pp. 83:1–83:8, ACM, 2016.

[4] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Computer Aided Verification* (T. Touili, B. Cook, and P. Jackson, eds.), pp. 24–40, Springer Berlin Heidelberg, 2010.

[5] K. Nepal, Y. Li, R. I. Bahar, and S. Reda, "ABACUS: A technique for automated behavioral synthesis of approximate computing circuits," in *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '14, pp. 361:1–361:6, IEEE, 2014.

[6] G. Liu and Z. Zhang, "Statistically certified approximate logic synthesis.," in *Proceedings of the 36th International Conference on Computer-Aided Design*, ICCAD '17, pp. 344–351, IEEE, 2017.

[7] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan, "ASLAN: Synthesis of approximate sequential circuits," in *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '14, pp. 364:1–364:6, IEEE, 2014.

[8] A. Chandrasekharan, M. Soeken, D. Große, and R. Drechsler, "Precise error determination of approximated components in sequential circuits with model checking," in *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, pp. 1–6, ACM, 2016.

[9] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *Proceedings of the 52nd Annual Design Automation Conference*, DAC '15, pp. 86:1–86:6, ACM, 2015.

[10] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "EvoApproxSb: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '17, pp. 258–261, IEEE, 2017.

[11] U. Meyer-Baese, *Digital signal processing with field programmable gate arrays*. Springer, 2014.

[12] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose, and V. Betz, "VTR 7.0: Next generation architecture and CAD system for FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, pp. 6:1–6:30, July 2014.

[13] D. Lundgren, "OpenCores jpegencode." https://github.com/chiggs/oc_jpegencode. Accessed: 2017-07-25.