ApproxFPGAs: Embracing ASIC-Based Approximate Arithmetic Components for FPGA-Based Systems

Bharath Srinivas Prabakaran∗,†, Vojtech Mrazek†,‡, Zdenek Vasecek‡, Lukas Sekanina†, Muhammad Shafique∗
∗Institute of Computer Engineering, Technische Universität Wien (TU Wien), Austria
†Faculty of Information Technology, IT4Innovations Centre of Excellence, Brno University of Technology, Czech Republic
‡Faculty of Information Technology, IT4Innovations Centre of Excellence, Brno University of Technology, Czech Republic

Abstract—There has been abundant research on the development of Approximate Circuits (ACs) for ASICs. However, previous studies have illustrated that ASIC-based ACs offer asymmetrical gains in FPGA-based accelerators. Therefore, an AC that might be pareto-optimal for ASICs might not be pareto-optimal for FPGAs. In this work, we present the ApproxFPGAs methodology that uses machine learning models to reduce the exploration time for analyzing the state-of-the-art ASIC-based ACs to determine the set of pareto-optimal FPGA-based ACs. 
We also perform a case-study to illustrate the benefits obtained by deploying these pareto-optimal FPGA-based ACs in a state-of-the-art automation framework to systematically generate pareto-optimal approximate accelerators that can be deployed in FPGA-based systems to achieve high performance or low-power consumption.

Index Terms—Approximate Computing, FPGA, ASIC, Adder, Multiplier, Arithmetic Units, Machine Learning, Statistics, Models, Synthesis.

I. INTRODUCTION

Field Programmable Gate Arrays (FPGAs) have become increasingly popular since their introduction in 1984 [1]. Due to their (partial) run-time reconﬁgurability, short time-to-market, and lower prototype costs, as compared to Application-Speciﬁc Integrated Circuits (ASICs), FPGAs are preferred in a wide variety of applications. These comprise domains like high-performance computing clusters and server platforms that offer “FPGAs as a Service”, and embedded and cyber-physical systems, which perform complex data-computations on the conﬁgurable arrays [2]. The current generation of FPGAs are equipped with a wide range of capabilities that can be used to design a Programmable System (on a Chip) by including hard IPs (IC realization) of the low-power ARM A9 processor core and other commonly used hardware accelerators, such as video codecs [3]. However, FPGAs are low-performance, power-hungry devices that are a lot less energy-efﬁcient when compared to ASICs.

The Approximate Computing paradigm offers a direction of research, in which the intermediate computational units can be approximated without “signiﬁcantly” degrading the output quality, to obtain savings in power/energy consumption and latency [4]. This quality of error-tolerance is exhibited by applications in the ﬁelds of recognition, mining, and synthesis, due to the following four factors: (i) redundancy in the processed data, (ii) algorithms with error attenuating patterns, (iii) non-existence of a unique golden output, and (iv) imperceptible differences in the output quality by end-users. Since its re-emergence, plenty of research works from academia and industry have exploited this phenomenon across the hardware [5]–[16] and software [17]–[20] layers to obtain power/energy/latency savings.

Most of the current works on approximate circuits (AC) primarily focus on obtaining energy/power/latency savings in ASIC-based systems. Previous studies have illustrated that ASIC-based approximate computing principles and techniques offer asymmetric savings when implemented on FPGAs [13] [15] [16]. State-of-the-art ACs for adders and multipliers can offer up to 70% savings in energy when synthesized for ASICs. Whereas these designs offer minimal/asymmetric savings or at times negative savings, i.e., an increase in resources when synthesized for FPGAs. This is primarily due to the architectural differences between ASICs and FPGAs. The required functionality is realized using logic gates in ASICs and using Lookup Tables (LUTs) made of SRAM elements in FPGAs. Therefore, an AC that offers significant savings and introduces the least error (pareto-optimal) for ASICs, might not necessarily be pareto-optimal for FPGAs. Note, by pareto-optimal approximate circuits we mean the set of all circuits that are not dominated by any other circuit from the set of circuits in the library in terms of the evaluation metrics.

Furthermore, the works presented in [13]–[16] have developed FPGA-based approximate circuits by analyzing the architecture of the target FPGA. These techniques are typically not scalable, due to their manual lookup table optimizations and approximations, and do not offer multiple pareto-optimal design points that trade-off between power consumption and introduced error. To further illustrate these behavioral differences between ASICs and FPGAs, we present a motivational analysis of our work in the next sub-section.

A. Motivational Analysis

We synthesize and implement a small subset of 4,494 8x8 unsigned approximate multiplier designs from the library of evolutionary approximate arithmetic circuits [21] and the state-of-the-art FPGA-based approximate multiplier designs [16]. These circuits were synthesized and implemented for the Xil-
**Fig. 1: Analysis of Pareto-optimal Approximate Circuits for Approximate 8x8 Multipliers and State-of-the-Art (SoA) FPGA-Based Approximate Multipliers [16].**
in the target application. Without loss of generality, in this work, we consider the evolutionary library of approximate adder and multiplier circuits for illustrating the benefits of our methodology [21]. Note, the use of other state-of-the-art designs is orthogonal to our approach and they can be appropriately included, with necessary modifications, in the library of approximate circuits.

**Exhaustive Exploration:** Due to the large number of designs present in the library, the time required for exploring all the designs exhaustively, might be quite large, as initially stated in Section I. Fig. 3 presents a brief illustration of the estimated time required for synthesizing all the approximate circuits present in the library for the target FPGA. As can be observed, when the number of ACs in the library increases, the time required for exploring the designs rises and reaches a magnitude of 100s of hours. Therefore, exhaustive exploration is not a feasible option for identifying the pareto-optimal approximate circuits for FPGAs. Fig. 3 also illustrates the savings in exploration time when the proposed ApproxFPGAs methodology is used for exploration as opposed to exhaustive exploration. The exploration time is reduced by a factor of $\sim 10\times$ from 82.4 days to 8.2 days, including the time required for synthesizing the data-set, training and evaluating the ML models, and re-synthesizing the pareto-optimal FPGA-ACs.

**ML-Model Learning:** Due to the infeasible time requirements of exhaustive exploration, we propose to train and evaluate a wide variety of statistical and machine learning (S/ML) models, which can be used to estimate the resource requirements of an approximate circuit, given its hardware description. These S/ML models can be used to estimate FPGA parameters like power consumption ($W$), latency ($ns$), and area ($\#LUTs$). Training these models requires a labeled data-set, with the FPGA parameters as the output labels and the hardware description of the AC as the input data. We build this data-set by randomly extracting a 10% subset of the complete library of ACs and synthesizing them for the target FPGA platform. This subset is further partitioned into training (80%) and validation (20%) data-sets, which are then used to train and evaluate the various machine learning models, respectively. Without loss of generality, in this work, we evaluate the applicability of the most-commonly used light-weight S/ML models (see Table I) to reduce the time required for exploring the library of ACs. We iteratively evaluate the accuracy of the models and modify their parameters based on the correlation obtained on the validation data-set to further improve the model’s accuracy. Instead of synthesizing and implementing each circuit in the library, which might take weeks to months, we can roughly estimate the FPGA parameters of all circuits using these models in the order of seconds. To estimate the accuracy of these ML models, we propose the *fidelity* metric, which evaluates the relationship between the measured ($mes$) and estimated ($est$) FPGA parameters for any two ACs in the library. We compute the *fidelity* ($F$) of a set of ACs, $X$, as:

$$F(X) = \frac{\sum_{x_1 \in X} \sum_{x_2 \in X} E(x_1, x_2)}{|X|^2}$$

(1)

where $E$ denotes the correctness of the relationship between the estimated and measured FPGA parameters:

$$E(x, y) = \begin{cases} 1 & \text{if } \text{est}(x) \mathcal{R} \text{est}(y) \land \text{mes}(x) \mathcal{R} \text{mes}(y) \\ 0 & \text{otherwise} \end{cases}$$

(2)

where $\mathcal{R}$ denotes one of the following relations $\{<, >, =\}$ between the FPGA parameters of the ACs. Due to their availability

**Pareto Construction:** Based on the outcome of our experiments (see Section IV), we select the best S/ML models to estimate the FPGA parameters of all ACs in the library. Based on these parameter-estimates, we can determine the

![Fig. 2: An Overview of the ApproxFPGAs Methodology](image)

![Fig. 3: Time Required for Exhaustive Exploration Compared to our ApproxFPGAs Approach for all ACs in the Library.](image)

**TABLE I: List of Light-weight Statistical/Machine Learning Models Used in ApproxFPGAs.**

| Statistical/ML Model | ML1 | ML2 | ML3 | ML4 | ML5 | ML6 | ML7 | ML8 | ML9 |
|----------------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| ML1 Regression w.r.t. ASIC-AC Power | Regression w.r.t. ASIC-AC Latency | Regression w.r.t. ASIC-AC Area | PLS Regression | Random Forest | Gradient Boosting | Adaptive Boosting (AdaBoost) | Gaussian Process | Symbolic Regression |
| ML10 Kernel Ridge | ML11 Bayesian Ridge | ML12 Coordinate Descent (Lasso) | ML13 Least Angle Regression | ML14 Ridge Regression | ML15 Stochastic Gradient Descent | ML16 K-Nearest Neighbours | ML17 Multi-Layer Perceptron (MLP) | ML18 Decision Tree |
pareto-optimal FPGA-ACs. However, we have observed that these models have limited fidelity, because of which the real pareto-optimal ACs can be dominated by the ACs where the estimation was incorrect. Therefore we propose to construct multiple pseudo-pareto-fronts from the input set (library) of ACs $C$. We determine the first set of pseudo-pareto-optimal ACs ($F_1$) from the initial set of all ACs $C$. Next, we eliminate all these pseudo-pareto-points from the input set to construct the second pseudo-pareto-front, i.e., using $C \setminus F_1$ as the input, we determine $F_2$. Similarly, we construct the third pseudo-pareto-front $F_3$, using the input $C \setminus (F_1 \cup F_2)$, and so on. By constructing multiple pseudo-pareto-fronts, we mitigate the inaccuracies associated with our S/ML models. The ACs lying on these pseudo-pareto-fronts can be subsequently synthesized again using our work-flow to determine the accurate FPGA parameters and the resources required. Hence, we have to synthesize an additional number of ACs when we are constructing multiple pseudo-pareto-fronts.

Based on the real FPGA parameter measurements obtained from the synthesis and implementation reports of Vivado, we construct an open-source library of pareto-optimal FPGA-ACs that offers a trade-off between the output quality and the resources consumed. This library can be subsequently utilized by application and system developers, to further maximize performance or power and energy savings obtained while satisfying the quality constraints of the application. The RTL and behavioral models of the FPGA-ACs are open-source and available online at https://github.com/ehw-fit/approx-fpgas.

**AutoAx-FPGA:** To incorporate the set of pareto-optimal FPGA-ACs in different error-tolerant applications, we modify the state-of-the-art AutoAx [23] framework to include the functionality of designing ACs for a given application that can be deployed in FPGA-based systems. The traditional AutoAx framework searches the design-space of approximate components to select and combine approximation components, in order to generate an approximate hardware accelerator that maximizes the energy savings. Initially, a set of random approximation assignments are evaluated for the target accelerator circuit, to get the quality of results (QoR) and hardware (HW) cost of the accelerator. Based on these values, QoR and HW cost estimators are constructed, which can be used to explore the complete design-space of approximate components for the given accelerator and to determine the set of pareto-optimal circuits for the given application. To generate approximate accelerators for a given application, which can be used in low-power and/or high-performance FPGA-based systems, we propose to include the following functionality in AutoAx: (i) we replace the library of pareto-optimal ASIC-ACs with the set of pareto-optimal FPGA-ACs obtained from the proposed ApproxFPGAs methodology, (ii) we modify the estimators used in AutoAx to estimate the FPGA parameters of the approximate accelerator instead of their ASIC-based HW costs.

### III. Experimental Setup

The RTL (in Verilog) and behavioral models (in C) of the evolutionary approximate arithmetic circuits are open-source and readily accessible\(^1\). These designs are synthesized and implemented (i.e., place & route) using the Vivado Design Suite 2017.2 for the target FPGA xc7vx485tffg1157-1, to extract their area, power, and timing reports. We restrict the placement and routing algorithms of the Xilinx Vivado by disabling the use of the FPGA’s DSP logic blocks. We do this to ensure that the designs are mapped on to the configurable logic. These reports are used to extract the FPGA parameters, which are subsequently used for training and evaluating the S/ML models. The S/ML models are implemented, trained, and tested inside the Python 3.7 environment with the help of the scikit-learn library. The RTL designs were synthesized on an Intel Core i5 – 7600 CPU with 16GB of internal memory and a 256GB Solid-State Drive (SSD). The S/ML models were trained and evaluated on an Intel Xeon CPU E5 – 2630 with 16GB of internal memory. An overview of our work-flow is presented in Fig. 4.

![Overview of Our Experimental Work-flow](image)

**Fig. 4:** Overview of Our Experimental Work-flow.

### IV. Results & Discussion

**Fidelity:** First, we illustrate the accuracy of the 18 S/ML models that we have evaluated inside our ApproxFPGAs framework. We do this by studying the fidelity of these models with respect to the three important FPGA parameters, namely, latency (ns), power (mW), and area (#LUTs). The fidelity of these models is evaluated on the validation data-set. The results of these experiments are presented in Fig. 5. From these results, we make the following key observations:

- Tree-based methods, like Decision Trees and Random Forest, achieve above-average accuracy in estimating the FPGA parameters and retaining their relationship to the other ACs.
- Based on further analysis, we also observed that generalization of models across all bit-widths is not very effective, i.e., estimating FPGA parameters of higher bit-width (12-/16-bit) designs (adder or multiplier) using a model learned from a lower bit-width (8-bit) designs is not very effective. On average, we observed that the fidelity of the higher bit-width designs decreased from 88% to 53% when using models

\(^1\)https://github.com/ehw-fit/evoapproxlib
trained with lower bit-width designs as opposed to designs of the same bit-width.

- Ridge models such as Kernel Ridge and Bayesian Ridge, typically, illustrate the best fidelity.

We also summarize the top-3 S/ML models for each FPGA parameter, along with the fidelity achieved for each case, in Table II. Likewise, we identify the models that achieve maximum fidelity when obtained using regression analysis on their corresponding ASIC parameters.

**TABLE II: Fidelity of the top-3 ML Models for the Estimating the FPGA parameters**

| Model | FPGA Latency | Model | FPGA Power | Model | FPGA Area |
|-------|--------------|-------|------------|-------|-----------|
| ML11  | 90%          | ML11  | 91%        | ML4   | 89%       |
| ML4   | 89%          | ML13  | 91%        | ML13  | 88%       |
| ML10  | 87%          | ML4   | 89%        | ML11  | 86%       |
| ML2   | 89%          | ML1   | 90%        | ML3   | 84%       |

**Correlation of ML Models:** Next, we illustrate the correlation between the estimated FPGA parameters and their measured values when the top-3 S/ML models are used on the library of approximate 16x16 multipliers. These results are illustrated in Fig. 6. From these results, we make the following key observations:

- The Bayesian Ridge and PLS regression techniques can be used as standalone techniques to estimate all three FPGA parameters, as they are one of the top-3 models for all three parameters.
- Statistical regression with respect to the corresponding ASIC parameters is equally useful in estimating the FPGA parameters of the given circuit.
- Due to the ~30% bias illustrated by the model, latency is not estimated accurately, especially using regression with ASIC parameters and Kernel Ridge. This leads to a scenario where the circuit latency is underestimated by the model, including certain pareto-optimal designs.

**Construction of the Pareto-fronts:** As discussed earlier, we construct multiple pareto-fronts to ensure that non-pareto-optimal designs are not missed by our methodology. Towards this, we illustrate the benefits of constructing multiple pareto-fronts sequentially for estimating the FPGA latency using the top-3 S/ML models and Regression with respect to ASIC latency. Fig. 7 illustrates the results of constructing 1, 2, and 3 pareto-fronts using the technique discussed in Section II. From these results, we make the following key observations:

- Using ML-based techniques for estimating the FPGA parameters reduces the total number of synthesized circuits by a factor of ~9× to 4,548, including the training and validation data-set and synthesis of pseudo-pareto-optimal points, instead of synthesizing the complete library of approximate 8x8 multipliers.
- The ML models are highly effective in selecting the pseudo-pareto-optimal designs that have to be re-synthesized, as compared to the regression analysis w.r.t ASIC latency, which increases the number of circuits to be explored from 79 in Bayesian Ridge to 164, effectively doubling the number of new circuits to be synthesized.
- The best results are obtained when we effectively combine the pseudo-pareto-optimal points obtained from multiple ML models. Therefore, we need to consider a union of all the pareto-fronts \( \bigcup_{i=1}^{n} F_i \) to determine the final set of pareto-optimal FPGA-ACs.

**Pareto-Optimal FPGA-ACs:** Fig. 8 illustrates the set of FPGA-ACs synthesized to obtain the subset of pareto-optimal FPGA-ACs using our proposed ApproxFPGAs methodology on the library of 8-, 16-bit adders and 8x8, 16x16 multipliers. Although we have not exclusively determined and synthesized...
all the pareto-optimal designs, we have reduced the exploration
time a factor of $\sim 10 \times$ to obtain, on average, 71% percentage
coverage of the pareto-optimal designs present in the library
of approximate circuits. This is quite explicitly illustrated with
the help of the pareto-front in the designs with a higher
number of ACs present in the library, such as the approximate
multipliers, and a little less explicit for libraries with a lower
number of circuits, like the approximate adders. Similarly,
we have generated the pareto-optimal ACs for the
12-bit approximate adder and 12x12 approximate multiplier.

**AutoAx-FPGA:** Finally, we present the results of modifying
the AutoAx framework to include the functionality of generating
pareto-optimal accelerators for FPGA-based systems.
We evaluated the modified AutoAx-FPGA methodology using
a Gaussian Filter as a case-study and the input of 9 pareto-
optimal 8x8 approximate multipliers and 8 16-bit approximate
adders. The QoR of the Gaussian filter’s output is estimated
using the structural similarity index (SSIM), for which we
build an estimator. First, we generate a training and validation
data-set of 5,000 random approximate circuits for the given
Gaussian filter, which was synthesized and implemented using
the Vivado work-flow to measure their FPGA parameters
such as area, latency, and power consumption. Similar to
the AutoAx methodology, we constructed estimators that can
determine the FPGA parameters for the other circuits in
the library, and construct 3 different pareto-fronts using the
hill-climber algorithm. We thereby reduce the number of
accelerator circuits to be explored from $4.95 \times 10^{14}$ to 368,
evaluate the models’ applicability. Based on the outcome, we circuits for FPGA-based systems. We synthesize a partial the use of current state-of-the-art ASIC-based approximate We presented the methodology, for embracing the use of approximate computing framework to illustrate the benefits obtained. This work was partially supported by Doctoral College Resilient Embedded Systems which is run jointly by TU Wien’s Faculty of Informatics and FH-Technikum Wien, and partially by Czech Science Foundation project 19-10137S. REFERENCES [1] S. M. S. Trimberger, “Three ages of fpgas: A retrospective on the first thirty years of fpga technology,” IEEE Solid-State Circuits Magazine, vol. 10, no. 2, pp. 16–29, 2018. [2] R. Watanabe et al., “Implementation of fpga building platform as a cloud service,” in Proceedings of the 10th HEART. ACM, 2019. [3] L. H. Crockett et al., The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc. Strathclyde Academic Media, 2014. [4] V. K. Chippa et al., “Analysis and characterization of inherent application resilience for approximate computing,” in DAC, ACM, 2013. [5] H. Jiang et al., “A comparative review and evaluation of approximate adders,” in GLSVLSI. ACM, 2015. [6] H. Jiang et al., “A comparative evaluation of approximate multipliers,” in NANOARCH. IEEE, 2016. [7] S. Mittal, “A survey of techniques for approximate computing,” ACM Computing Surveys (CSUR), vol. 48, no. 4, p. 62, 2016. [8] S. Hashemi et al., “Drum: A dynamic range unbiased multiplier for approximate applications,” in ICCAD. IEEE Press, 2015. [9] H. Saadat et al., “Approximate integer and floating-point dividers with near-zero error bias,” in DAC. ACM, 2019. [10] H. Saadat et al., “Minimally biased multipliers for approximate integer and floating-point multiplication,” IEEE TCAD, vol. 37, no. 11, pp. 2623–2635, 2018. [11] S. Venkataramani et al., “Quality programmable vector processors for approximate computing,” in MICRO. IEEE, 2013. [12] A. Sampson et al., “Enerj: Approximate data types for safe and general low-power computation,” in ACM SIGPLAN Notices, vol. 46, no. 6. ACM, 2011, pp. 164–174. [13] B. S. Prabakaran et al., “Demas: An efficient design methodology for building approximate adders for fpga-based systems,” in DATE. IEEE, 2018. [14] J. Echavarria et al., “Fau: Fast and error-optimized approximate adders on lut-based fpgas,” in FPT. IEEE, 2016. [15] S. Ullah et al., “Smapproxlib: library of fpga-based approximate multipliers,” in DAC. IEEE, 2018. [16] S. Ullah et al., “Area-optimized low-latency approximate multipliers for fpga-based hardware accelerators,” in DAC. ACM, 2018. [17] A. K. Mishra et al., “iact: A software-hardware framework for understanding the scope of approximate computing,” in WACAS. 2014. [18] W. Baek et al., “Green: a framework for supporting energy-conscious programming using controlled approximation,” in ACM Sigplan Notices, vol. 45, no. 6. ACM, 2010, pp. 198–209. [19] D. S. Khudia et al., “Rumba: An online quality management system for approximate computing,” in FSCA. IEEE, 2015. [20] A. Yazdanbakhsh et al., “Axilog: Language support for approximate hardware design,” in DATE. IEEE EDA Consortium, 2015. [21] V. Mrazek et al., “Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods,” in DATE. IEEE European Design and Automation Association, 2017. [22] J. Han et al., “Approximate computing: An emerging paradigm for energy-efficient design,” in ETS. IEEE, 2013. [23] V. Mrazek et al., “autoax: An automatic design space exploration and circuit building methodology utilizing libraries of approximate components,” in DAC. ACM, 2019.