SMApproxLib: Library of FPGA-based Approximate Multipliers

Salim Ullah, Sanjeev Sripadraj Murthy, Akash Kumar
Technische Universität Dresden, Germany
Corresponding Author's Email: salim.ullah@tu-dresden.de

ABSTRACT

The main focus of the existing approximate arithmetic circuits has been on ASIC-based designs. However, due to the architectural differences between ASICs and FPGAs, comparable performance gains cannot be achieved for FPGA-based systems by using the approximations defined, particularly for ASIC-based systems. This paper exploits the structure of the 6-input lookup tables and associated carry chains of modern FPGAs to define a methodology for designing approximate multipliers optimized for FPGA-based systems. Using our presented methodology, we present SMApproxLib, an open source library of approximate multipliers with different bit-widths, output accuracies and performance gains. Being the first open source library of FPGA-based approximate multipliers, SMApproxLib can serve as a benchmark for designing and comparing future FPGA-based approximate arithmetic circuits.

KEYWORDS

Approximate Computing, Multipliers, Adders, FPGAs, Optimization, Area, Latency, Design Space Exploration

1 INTRODUCTION

The reconfigurable nature of FPGAs has made them an attractive choice for a wide range of applications by reducing both the time-to-market, as well as the associated costs of developing new systems. In order to provide high performance for different applications, modern FPGAs also host hard DSP blocks. These DSP blocks are optimized to perform various fixed point and floating point operations such as multiplication and division. Synthesis tools tend to deploy these hard DSP blocks to reduce the overall execution time and power consumptions of different applications. However, as noted by Kuon et al. [6], the usage of DSP blocks might result in performance degradation of some applications. This performance degradation is mainly due to the fixed locations of the DSP blocks. Similarly, for some applications, the usage of DSP blocks might result in exhaustion of the DSP blocks in concurrently running applications. Figure 1 shows the implementation results of four applications using Xilinx Vivado 17.1 for Virtex-7 series FPGA[1]. As shown in Figure 1(a), the fixed location of DSP blocks has resulted in an increase in the critical path delays of Viterbi Decoder, Reed-Solomon Encoder and Image WARP applications. Similarly, the implementation of the Viterbi Decoder, in Figure 1(b), has resulted in the utilization of a very large number of available DSP blocks. Therefore, it is always desirable to have logic-based soft multipliers along with DSP blocks.

[1]The default Balanced Synthesis option was used.

This work was supported by the German Research Foundation(DFG) funded Project ReAp under Grant 380524764.
DAC '18, June 24–29, 2018, San Francisco, CA, USA

ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3196115

The works of [5, 10, 13] also concentrate on efficient utilization of FPGA resources for multiplier implementations. However, there are a range of applications that are resilient to inexactness of input-data or intermediate computations, and for such applications the accurate intermediate computations or exact final outputs may not be required to produce acceptable output quality. Examples of such applications can be found in the domain of media processing, data mining and recognition applications [2, 3]. Exploiting the error-resilience of these applications, logic-based approximate multipliers can be designed with better area, performance and energy efficiencies. However, most of the recent works on approximate arithmetic circuits, concentrate on either ASIC-based designs or do not consider the underlying structures of FPGAs, i.e. lookup tables (LUTs) and carry chains. The authors of [1, 7, 8] have used approximate partial product reduction trees to achieve performance and energy gains. [4, 11] have presented approximate 2×2 multipliers to generate higher order multipliers. By combining different approximate adders and multipliers present in literature, [9] presented "EvoApprox8b", an open source library of 8-bit approximate adders and multipliers. However, the EvoApprox8b concentrates on designing 8-bit approximate adders and multipliers. Moreover, due to the ignorance towards the architectural specifications of FPGAs, deployment of these approximate designs on FPGAs do not produce comparable performance, energy and area gains. Figure 2 presents the ASIC-based implementation results of four 8×8 multipliers, chosen randomly from EvoApprox8b [9], along with their FPGA-based implementations. These results show the performance gains of approximate multipliers - A1, A2, A3 and A4, with respect to an accurate multiplier, also obtained from EvoApprox8b. The results for ASIC-based implementations have been reported from [9]. For FPGA-based results, we have implemented A1—A4 on Virtex 7 FPGA using Xilinx Vivado 17.1. As shown in Figure 2, the performance gains of ASIC-based approximate designs are not comparably transported to the FPGA domain.
In order to provide a road map for obtaining area, energy and performance gains by using approximate computing for FPGA-based systems, this paper emphasizes on developing designs by considering the structure of modern FPGAs. To address the aforementioned problems, this paper presents *SMApproxLib* an open source Library of Approximate Soft Multipliers, optimized for modern FPGAs such as those provided by Xilinx and Intel.

Utilizing an FPGA-based accurate n×n multiplier design, optimized for area and energy efficiency, the **novel contributions** of this paper include a design space exploration methodology for generating approximate multipliers of arbitrary data sizes. For each n×n accurate multiplier, we provide three approximate n×n multiplier designs by efficient utilization of LUTs and carry chains. In order to reduce the execution time of an n×n multiplier, our methodology recommends implementing it using four instances of $\frac{n}{2}$ multipliers. Each individual instance of $\frac{n}{2}$ multiplier can be generated either directly or recursively from four instances of $\frac{n}{4}$ multipliers. Besides supporting accurate summation of partial products, we also provide a novel n-bit approximate adder to reduce the overall execution

time of the multiplier. All implementations have been characterized by their area and latency requirements and average relative errors. SMApproxLib incorporates the non-dominating design points offered by EvoApprox8b [9] to provide a rich library of multipliers optimized for FPGA-based systems. Being the first library to provide approximate multipliers optimized particularly for FPGAs, we will make our automated tool flow for generating the VHDL and behavioral codes, for implementing multipliers of arbitrary sizes, open-source at *https://cfaed.tu-dresden.de/pd-downloads*.

The rest of the paper is organized as follows: in Section 2, we briefly present the preliminaries required to understand the proposed methodology. Section 3 introduces an FPGA-based accurate multiplier architecture optimized for area and energy efficiency. The proposed approximate multiplier architectures are discussed in Section 4, followed by a discussion about our design space exploration and integration of design points from other sources in Section 5. Section 6 discusses the experimental setup, implementation of proposed designs and their results, followed by the Conclusion in Section 7.

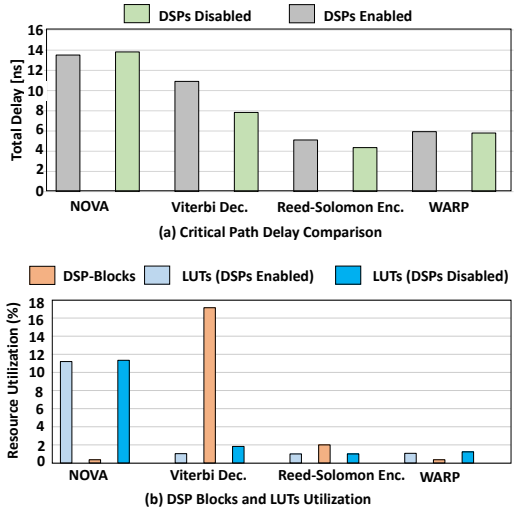

Figure 1: Effects of DSP utilization on the implementation results for four applications

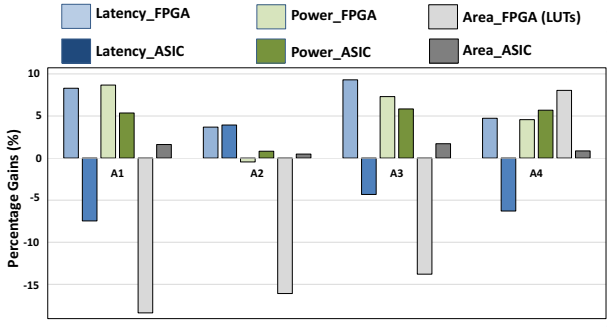

Figure 2: Implementation results of ASIC-optimized multipliers on FPGAs

2 PRELIMINARIES

The configurable logic blocks (CLBs) are the main resources involved in implementing any kind of sequential or combinatorial circuit on Xilinx FPGAs. Unlike previous versions of Xilinx FPGAs, a CLB in the state-of-the-art FPGAs has eight 6-input lookup tables (LUTs) and an 8-bit long carry chain. As shown in Figure 3, the 6-input LUTs can be used to implement either a single 6-input combinational function or two 5-input combinational functions by defining a 64-bit INIT value. The INIT value of an LUT describes all the possible combinations for which a "1" is received at the output of the LUT. The LUTs are also used to control the associated carry chain. In this paper, Xilinx Virtex 7-series FPGA has been used for the implementation of the proposed approximate multiplier library. In the 7-series FPGAs, a CLB has two slices to group eight LUTs and the associated carry chain, as shown in Figure 4.

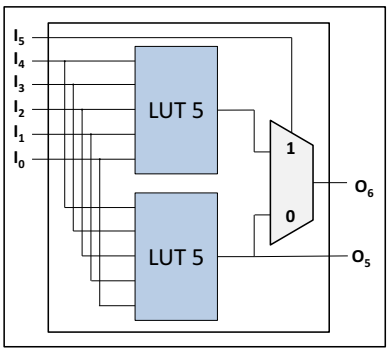

Figure 3: 6-input LUT of Xilinx FPGAs [15]

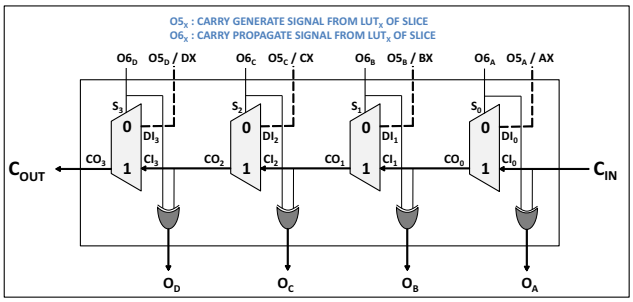

Figure 4: Carry chain of Xilinx FPGAs [15]

Unlike ASIC-based designs which provide a finer design control of the circuits at gate level granularity, FPGAs provide design control at LUT level of granularity. Therefore, any design optimized for FPGAs, must consider the structure of individual slices of FPGAs. Ignorance towards the structure and direct porting of ASIC-based designs result in loss in performance gains, as observed in our motivational case study in Section 1. Therefore, instead of completely relying on FPGA synthesis and implementation tools for optimizing the designs, our methodology provides a base architecture for accurate multiplication and then utilizes this structure to introduce different types of approximations.

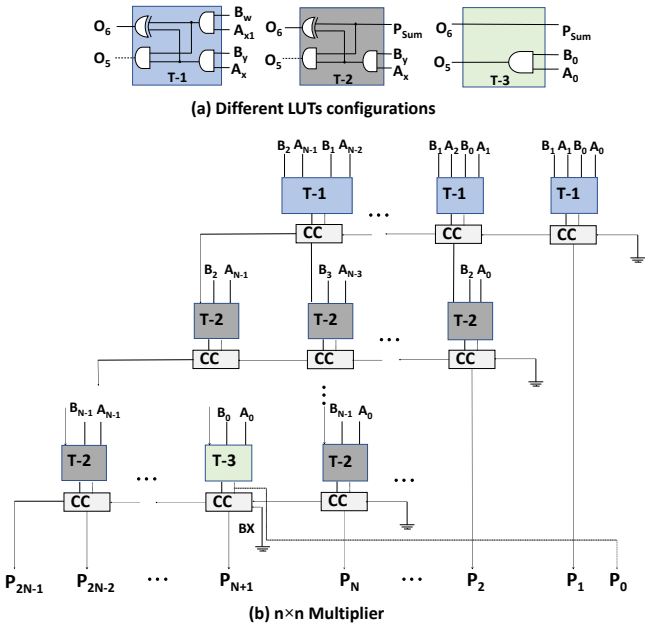

Figure 5: Area and energy efficient implementation of n×n multiplier

3 FPGA OPTIMIZED ACCURATE MULTIPLIER

Utilizing the general structure of an n×n multiplier and exploiting the 6-input LUTs and fast carry chains of modern FPGAs, an accurate $n \times n$ array multiplier has been shown in Figure 5. This accurate multiplier combines the processes of partial products generation and their accumulation into a single step, to efficiently utilize FPGA resources. For this purpose, it classifies the LUTs into three configurations: T-1, T-2 and T-3 as shown in Figure 5(a). As shown in Figure 5(b), the first row of this accurate multiplier has LUTs with T-1 configuration and it generates the first two rows of partial products of an n×n multiplier and accumulates them using the associated carry chain. The remaining $n-2$ rows have LUTs with configuration T-2 for generation of the remaining partial products and their accumulation through the carry chains. The last row has an LUT of Type T-3 for generation of product bits P_{N+1} and P_0. Using this accurate multiplier, we propose different approximation techniques to enable gains in area, latency and energy.

4 PROPOSED APPROXIMATE MODULES

To improve the performance of the implementation of the accurate multiplier shown in Figure 5, we propose three novel approximations. These approximations aim to reduce the critical path delay and power consumption, by reducing the number of logic levels present in Figure 5. For all approximate designs, the partial products are divided into multiple *layers*. To fully utilize the six inputs of an LUT, each *layer* contains four consecutive rows of partial products, as shown by the black line in Figure 6 for an 8×8 multiplier. The mapping of four consecutive partial products rows to 6-input LUTs is further explained in the following sections.

4.1 Approximate Design 1: Approx1

Approx1 uses approximate addition for the reduction of the partial products to a final answer. Utilizing LUTs of Type1 and associated carry chains from Figure 5, every two consecutive partial products rows are generated and mutually added accurately. This process is performed in parallel for all partial products. For example, the generation and summation of Row1 & Row2 and Row3 & Row4 in Figure 6 results in sum vectors ($S_{09} - S_{00}$) and ($S_{19} - S_{10}$) respectively. These sum vectors can be reduced to a final answer ($PP_{11} - PP_0$) by using either accurate addition through carry chains, or by utilizing the 6-input LUT based approximate addition, as shown in Figure 7. The INIT values (hex) in Figure 7 show the approximate addition performed by each LUT. To improve the accuracy of the addition, every LUT considers bit values of the preceding column, for calculating the current output bit. The same approximate addition can be used to obtain the final product by adding the final outputs of each *layer*. It can be seen that for an n×n multiplier, Approx1 is capable of generating and reducing partial products to compute the final product in $\lceil log_2(n) \rceil$ steps.

4.2 Approximate Design 2: Approx2

Approx2 and Approx3 are optimized to improve latency and energy gains, by removing the associated carry chains for partial products accumulation. As discussed previously, the first step in all approximate designs is to group the partial products into multiple *layers*, with each *layer* containing four rows of consecutive partial products. For Approx2 and Approx3, we have further grouped partial products in each *layer*, as shown by the different color boxes in Figure 8. The grouping is based on the location of a partial product bit with respect to other partial product bits in a *layer*. Approx2 and Approx3 differ only in the computation of partial product bits enclosed by the green boxes. As shown by the red box in Figure 8(a), which is a representation of Rows $1-4$ of Figure 6, the first two columns in each *layer* can be accurately produced by a single LUT. The corresponding red box in Figure 8(b) shows the implementation of the function performed by the LUT. Similarly, the blue box in Figure 8(b), utilizes a single LUT to accurately generate and add the partial products in the third column. The yellow box in third last column of each *layer*, is utilized for approximately computing the sum of partial products. The approximation in the function performed by the yellow box is the result of predicting the carry-out from the preceding column. The purple boxes are utilized to compute the final two product bits in each *layer*. As shown in Figure 8(b), the purple boxes utilize two LUTs for improving the accuracy of the most significant bit. As all the partial product bits grouped by green boxes contain four partial product terms, the proposed approximation utilizes two 6-input LUTs per group for generating and computing the sum of these bits. As shown by Figure 9(a), Approx2 utilizes a 6-input LUT for generating the first two partial product terms and then adds them by using an XOR gate. This generates the approximate sum of the first two partial product terms. As can be seen in Figure 9(a), a second LUT is utilized for generating the next two partial product terms and adds them with the previously computed sum. The same technique is used for all partial product bits that are enclosed by green boxes, in each *layer*. Since the generation and summation of all partial products bits in every group is executed in parallel, Approx2 offers reduced latency as compared to Approx1 and the accurate multiplier presented in Section 3. The elimination of the carry chain also results in the reduction of energy consumption in the circuit.

4.3 Approximate Design 3: Approx3

Approx3 is different from Approx2 in the computation of partial product terms enclosed by green boxes. In order to improve the

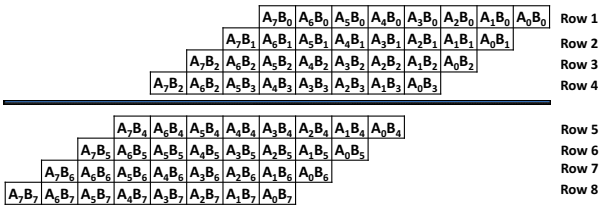

Figure 6: 8×8 array multiplier

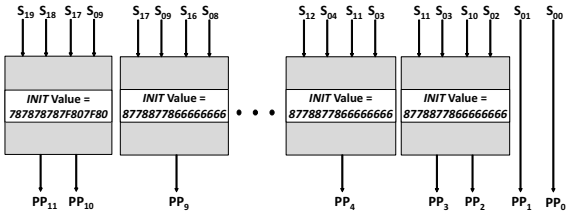

Figure 7: Approximate addition of partial products

accuracy of Approx2, Approx3 considers predicting the carry out from preceding bit locations. As shown by Figure 9(b), a 6-input LUT is used for computing the first three partial product terms in each group. This LUT also implements three AND gates, identified by shading, to predict the carry from preceding bit locations. The computed three partial product terms and the predicted carry are added together to generate an approximate sum. As shown in Figure. 9(b), by using a second LUT, this sum is again added with the fourth partial product term to compute the final approximate sum of the group.

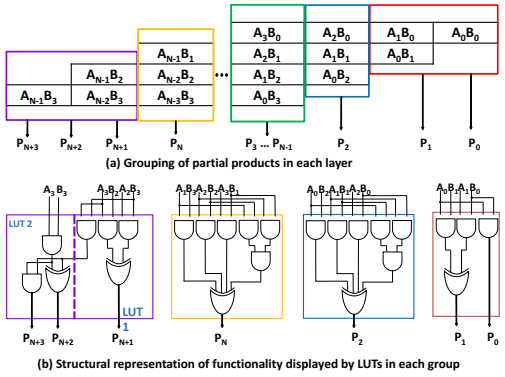

Figure 8: Partial products generation for Approx2 and Approx3

5 DESIGN SPACE EXPLORATION

Using the accurate multiplier, approximate adder and the three approximate multiplier designs, we propose a design space exploration methodology and provide an open source automated tool flow for implementing approximate multipliers of arbitrary sizes, with different performance gains. Our methodology performs mapping of logic to LUTs at design time and allows for quick estimation of area and latency requirements of the design. The advantage of

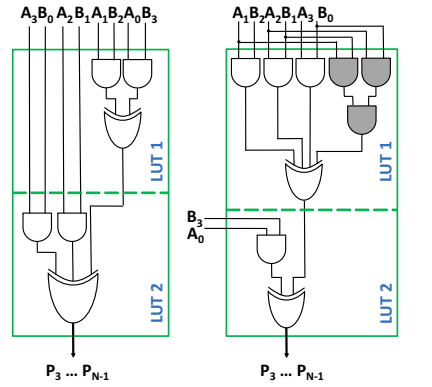

Figure 9: Structural difference between Approx2 and Approx3

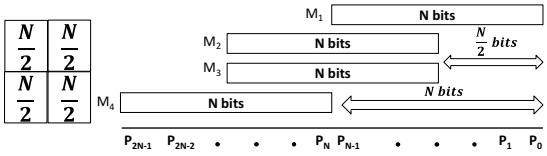

Figure 10: Recursive approach for higher order multipliers

design time mapping of logic to LUTs is that similar performance gains can be obtained for different versions of FPGAs and synthesis tools. Besides providing array-based accurate and approximate multipliers, our methodology supports a recursive approach of designing $N \times N$ multipliers from $\frac{N}{2}$ multipliers as shown in Figure 10. By using the recursive approach, area/latency requirements and the worst case error of an $N \times N$ multiplier can be estimated from Eq.1–Eq.3.

$$LUTs\ for\ (N \times N)\ multiplier = \sum_{i=1}^{4} (LUTs\ for (\frac{N}{2} \times \frac{N}{2})\ multiplier\ i) + LUTs\ for\ Adder \quad (1)$$

$$Latency\ of\ (N \times N) multiplier \approx max(Latency for(\frac{N}{2} \times \frac{N}{2}) multiplier) + Latency for Adder \quad (2)$$

$$Worst\ case\ error of (N \times N) multiplier \approx Worst\ case\ error\ of\ M_1 + Worst\ case\ error\ of\ M_2 \times 2^N + Worst\ case\ error\ of\ M_3 \times 2^N + Worst\ case\ error\ of\ M_4 \times 2^2 N \quad (3)$$

Since the proposed methodology utilizes both approximate partial product generation and approximate summation of partial products, it is also capable of integrating other approximate designs to provide a wide range of design choices for different applications.

Table 1: Area, latency, power and error values of proposed multipliers

Performance Metric	4x4					8x8				
	Accurate	Approx 1	Approx 2	Approx 3	Xilinx Multiplier IP	Accurate	Approx 1	Approx 2	Approx 3	Xilinx Multiplier IP
Area (LUTs)	12	12	7	7	16	56	64	54	54	64
Latency (ns)	6.796	6.094	5.254	5.531	6.027	10.592	9.222	9.038	9.081	8.417
Power (W)	0.198	0.18	0.192	0.192	0.245	0.22	0.22	0.217	0.216	0.344
Average Relative Error	0.0	0.072	0.126	0.123	0.0	0.0	0.016	0.030	0.027	0.0

6 RESULTS AND DISCUSSION

We have implemented an automated tool flow for generating the VHDL and behavioral (MATLAB) codes of all proposed multipliers. Our automated tool flow utilizes Vivado 17.1 for the synthesis and implementation of the proposed multipliers for the 7VX330T device of Virtex-7 family. Vivado simulator and Power Analyzer have been utilized for calculation of the power values. For power reports, a uniform distribution based on all combinations of input values has been used. Using the proposed methodology, 545 different implementations of an 8×8 multiplier have been produced, which are compared with EvoApprox8b [9] and IpACLib [12] for performance gains and resource utilization. The MATLAB codes have been utilized to analyze the effects of approximate multipliers on the final output quality of a Gaussian noise removal filter.

Table 1 presents the area, latency, power consumption and average relative error of four multipliers using the proposed methodology, along with those of the default multiplier IP provided by Xilinx [14]. These results demonstrate the proposed basic accurate and approximate architectures. The complete design space using these basic architectures is presented in Figure 12 – Figure 15. As shown in Table 1, all approximate designs trade accuracy for gains in latency and power consumption. Due to a separate partial product reduction tree using carry chains, the 8×8 Approx1 multiplier has higher area than the accurate 8×8 multiplier. However, the 8×8 Approx1 multiplier computes the final product in $log_2(8)$ steps and offers gains in latency and power requirements. Since Approx3 predicts previous carries for computing new product bits, its average relative error is lesser than that of Approx2. Eq.4 has been used for computing the average relative error. In order to check the effects of these approximate multipliers on the final output qualities of real world applications, we have implemented the Gaussian noise removal filter using our accurate and approximate multipliers. As shown by Figure 11 and Table 2, the proposed approximate multipliers have resulted in insignificant loss in output quality.

$$Average\ relative\ error = \frac{1}{N}\sum_{i=1}^{N} \left| \frac{Accurate\ result_i - Approximate\ result_i}{Accurate\ result_i} \right| \quad (4)$$

Further, we present a design space by integrating the design points from EvoApprox8b [9] and IpACLib [12]. As stated in the motivational analysis in Section 1, ignorance towards the underlying architecture of FPGAs has resulted in significant decrease in Pareto points for EvoApprox8b. By integrating the resultant

Table 2: PSNR values of the images in Figure 11

8x8 Multiplier	PSNR (dB)
Accurate	28.4335
Approx 1	28.2455
Approx 2	27.5388
Approx 3	27.2070

Pareto points of EvoApprox8b with design points offered by our proposed approximate multipliers, we present a wide-range design space for FPGA-based systems. By comparing the critical path delay and average relative errors of all design points, Figure 12 shows that EvoApprox8b offers 12 non-dominated design points, whereas 24 non-dominated points are presented by the proposed approximate multipliers. Similarly, Figure 13 presents the integrated design space for comparison of area and latency requirements of all design points, in which both, EvoApprox8b and the proposed approximate multipliers, offer 3 Pareto points. For power and area requirements of the design space, Figure 14 shows that the proposed methodology offers 3 non-dominated design points while 1 non-dominated point is provided by EvoApprox8b. Finally, Figure 15 presents the average relative error and area requirements of all the design points. The proposed methodology offers 15 non-dominated design points, whereas 23 non-dominated design points are provided by EvoApprox8b. In all of these comparisons, IpACLib [12] was filtered out due to it high area, latency and power consumption requirements.

7 CONCLUSION

In this paper, we presented a novel design methodology for designing approximate multipliers, optimized specifically for FPGA-based systems. By utilizing the 6-input LUTs and carry chains, we have proposed three approximate multiplier designs. By integrating the design points from EvoApprox8b, our methodology provides a rich library of approximate multipliers with various performance attributes. Since the proposed methodology is based on effective utilization of LUTs, our library is scalable to FPGAs from other vendors such as Intel. Besides these, we provide an automated tool flow for implementing accurate/approximate multipliers of arbitrary sizes with different performance gains. SMApproxLib, along with our automated tool flow will be made open-source at *https://cfaed.tu-dresden.de/pd-downloads* to encourage and help further research in this direction.

REFERENCES

[1] Kartikeya Bhardwaj, Pravin S Mane, and Jorg Henkel. 2014. Power-and area-efficient Approximate Wallace Tree Multiplier for error-resilient systems. In

(a) Accurate (b) Approx1 (c) Approx2 (d) Approx3

Figure 11: Gaussian noise removal filter output images: (a) with accurate multiplier, (b) with Approx1 multiplier, (c) with Approx2 multiplier and (d) with Approx3 multiplier

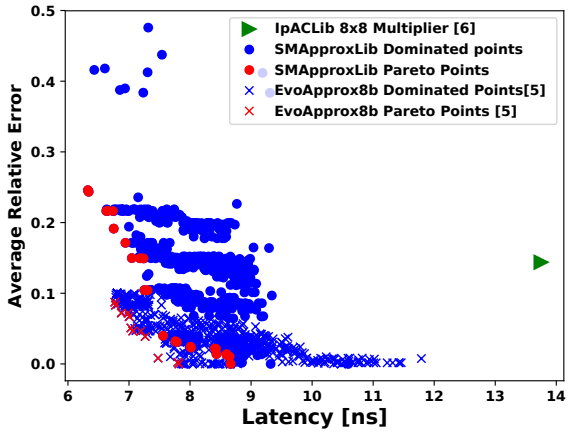

Figure 12: Latency and average relative error value comparison of proposed methodology with EvoApprox8b and IpACLib

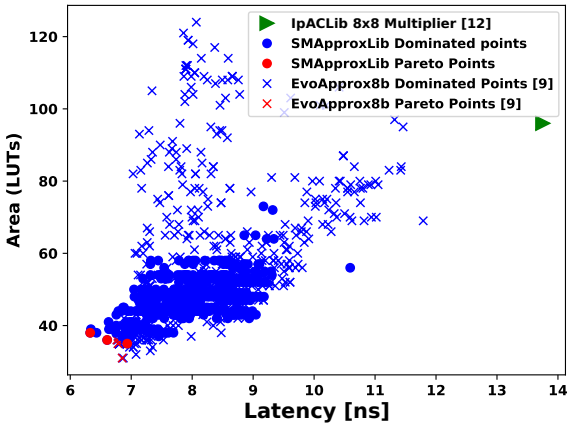

Figure 13: Latency and area value comparison of proposed methodology with EvoApprox8b and IpACLib

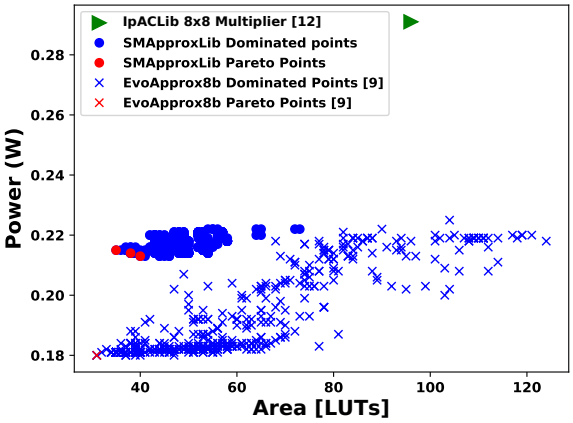

Figure 14: Power and area requirement comparison of proposed methodology with EvoApprox8b and IpACLib

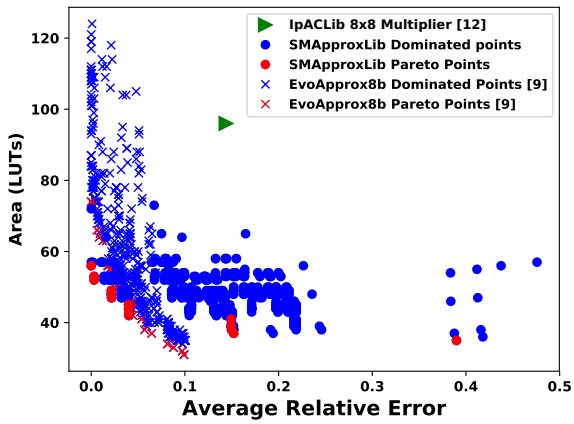

Figure 15: Area and average relative error value comparison of proposed methodology with EvoApprox8b and IpACLib

Quality Electronic Design (ISQED), 2014 15th International Symposium on. IEEE, 263–269.

[2] Vinay K Chippa, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. 2013. Analysis and characterization of inherent application resilience for approximate computing. In *Proceedings of the 50th Annual Design Automation Conference*. ACM, 113.

[3] Jie Han and Michael Orshansky. 2013. Approximate computing: An emerging paradigm for energy-efficient design. In *Test Symposium (ETS), 2013 18th IEEE European*. IEEE, 1–6.

[4] Parag Kulkarni, Puneet Gupta, and Milos Ercegovac. 2011. Trading accuracy for power with an underdesigned multiplier architecture. In *VLSI Design (VLSI Design), 2011 24th International Conference on*. IEEE, 346–351.

[5] Martin Kumm, Shahid Abbas, and Peter Zipf. 2015. An efficient softcore multiplier architecture for Xilinx FPGAs. In *Computer Arithmetic (ARITH), 2015 IEEE 22nd Symposium on*. IEEE, 18–25.

[6] Ian Kuon and Jonathan Rose. 2007. Measuring the gap between FPGAs and ASICs. *IEEE Transactions on computer-aided design of integrated circuits and systems* 26, 2 (2007), 203–215.

[7] Chia-Hao Lin and Chao Lin. 2013. High accuracy approximate multiplier with error correction. In *Computer Design (ICCD), 2013 IEEE 31st International Conference on*. IEEE, 33–38.

[8] Cong Liu, Jie Han, and Fabrizio Lombardi. 2014. A low-power, high-performance approximate multiplier with configurable partial error recovery. In *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 95.

[9] Vojtech Mrazek, Radek Hrbacek, Zdenek Vasicek, and Lukas Sekanina. 2017. EvoApproxSb: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 258–261.

[10] Hadi Parandeh-Afshar and Paolo Ienne. 2011. Measuring and reducing the performance gap between embedded and soft multipliers on FPGAs. In *Field Programmable Logic and Applications (FPL), 2011 International Conference on*. IEEE, 225–231.

[11] Semeen Rehman, Walaa El-Harouni, Muhammad Shafique, Akash Kumar, and Jörg Henkel. 2016. Architectural-space Exploration of Approximate Multipliers. In *Proceedings of the 35th International Conference on Computer-Aided Design (ICCAD '16)*. ACM, New York, NY, USA, Article 80, 8 pages. https://doi.org/10.1145/2966986.2967005

[12] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel. 2015. A low latency generic accuracy configurable adder. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/2744769.2744778

[13] Zarrin Tasnim Sworna, Mubin Ul Haque, Hafiz Md Hasan Babu, Lafifa Jamal, and Ashis Kumer Biswas. 2017. An Efficient Design of an FPGA-Based Multiplier Using LUT Merging Theorem. In *VLSI (ISVLSI), 2017 IEEE Computer Society Annual Symposium on*. IEEE, 116–121.

[14] Xilinx. 2011. Logicore IP multiplier v11.2. https://www.xilinx.com/support/documentation/ip_documentation/mult_gen_ds255.pdf

[15] Xilinx. 2017. Xilinx 7 Series Configurable Logic Block. https://www.xilinx.com/support/documentation/ip_documentation/mult_gen_ds255.pdf