DeMAS: An Efficient Design Methodology for Building Approximate Adders for FPGA-Based Systems

Bharath Srinivas Prabakaran[1], Semeen Rehman[1], Muhammad Abdullah Hanif[1], Salim Ullah[2], Ghazal Mazaheri[3], Akash Kumar[2], and Muhammad Shafique[1]
[1]Vienna University of Technology (TU Wien), Austria
[2]Technische Universität Dresden, Germany
[3]University of California, Riverside, United States of America
Contact Persons' Email: bharath.prabakaran@tuwien.ac.at, muhammad.shafique@tuwien.ac.at

***Abstract*—The current state-of-the-art approximate adders are mostly ASIC-based, i.e., they focus solely on gate and/or transistor level approximations (e.g., through circuit simplification or truncation) to achieve area, latency, power and/or energy savings at the cost of accuracy loss. However, when these designs are synthesized for FPGA-based systems, they do not offer similar reductions in area, latency and power/energy due to the underlying architectural differences between ASICs and FPGAs. In this paper, we present a novel generic design methodology to synthesize and implement approximate adders for any FPGA-based system by considering the underlying resources and architectural differences. Using our methodology, we have designed, analyzed and presented eight different multi-bit adder architectures. Compared to the 16-bit accurate adder, our designs are successful in achieving area, latency and power-delay product gains of 50%, 38%, and 53%, respectively. We also compare our approximate adders to state-of-the-art approximate adders specialized for ASIC and FPGA fabrics and demonstrate the benefits of our approach. We will make the RTL and behavioral models of our and state-of-the-art designs open-source at https://sourceforge.net/projects/approxfpgas/ to further fuel the research and development in the FPGA community and to ensure reproducible research.**

***Index Terms*—Approximate Computing, FPGA, Adders, LUTs, Optimization, Design Flow, Efficiency, Area, Power, Performance, CAD.**

I. Introduction

FPGAs serve as an excellent platform for a wide range of applications from small-scale embedded devices to high-performance computing systems due to their short time-to-market, enhanced flexibility and run-time reconfigurability. However, despite supporting specialized hardware accelerators and co-processors, FPGA-based systems typically consume more power and/or energy, compared to their ASIC counterparts. Therefore, besides employing traditional energy-optimization techniques, there is a need for exploring new avenues in energy-efficient computing solely for FPGA-based systems. One such attractive trend is the *Approximate Computing* paradigm, which is re-emerging due to the breakdown of Moore's law and Dennard scaling, and the ever-increasing demand for high-performance and energy efficiency.

Approximate computing trades the accuracy and precision of intermediate or final computations to achieve significant gains in critical path delay, area, power and/or energy consumption. This trade-off becomes beneficial for applications exhibiting inherent application resilience [1], i.e., the ability to produce viable output despite some of its computations being inaccurate because of approximations. A wide range of applications like image and video processing, data mining, machine learning, etc., in the recognition, mining and synthesis domains exhibit this property. Existing approximate computing techniques and principles can be applied to different stages of the computing stack, ranging from logic and architectures at the hardware layer all the way up to compiler and programming language at the software layer [2]. There is an extensive amount of research related to approximations at both hardware and software layers [3], [4]. Voltage over-scaling [5], [6] and functional approximation [7] are the two major approximate computing knobs employed at the hardware level. Approximations at the software level can be classified into two major categories: (i) loop perforation, function approximation [8], [9] and (ii) programming language support [10], [11]. Approximations at the hardware level are focused on basic computation modules like adders [12]–[14] and multipliers [15], [16]. Research works like [17], [18] focus on modeling the error probability of the existing state-of-the-art ASIC-based adders and recursive multiplier architectures. More recent works focus on architecture-level approximations targeting application-specific domains like video processing [19] to achieve energy efficiency. Major industrial players like Intel, IBM and Microsoft have also explored the approximate computing paradigm, and have demonstrated case studies on the design of energy-efficient hardware and software systems using approximation techniques [20]–[22].

There has been a lot of research in the field of approximate computing, focusing mostly on ASIC-based systems. However, due to the underlying architectural differences between ASICs and FPGAs, these approximate computing principles are not directly applicable to FPGAs for achieving similar gains. In the following, we present a motivational analysis to elaborate on this source of inefficiency.

A. Motivational Analysis

In this section, we illustrate that the ASIC-based approximate computing principles and components, when synthesized and implemented on FPGAs, are not efficient in achieving proportional area, power, latency and/or energy reductions compared to when these components are synthesized for ASICs. We utilize state-of-the-art approximate adder architectures present in the open-source *lpAClib* [23] and *ApproxAdder* [24] libraries. These adder architectures have been synthesized for a Xilinx Virtex-7 FPGA using the Xilinx ISE 14.7 tool-flow. Fig. 1 illustrates the analysis for area (LUTs), delay and power-delay product of the state-of-the-art ASIC-based adders. The Add1, Add2 and Add4 are the ASIC-based approximate adder versions proposed in [12], these adders are obtained by truncating and modifying transistors present in a ripple carry adder circuit to achieve latency, area and energy gains. On the other hand, the "GeAr" adder focuses on increasing the performance at the cost of accuracy and area [13].

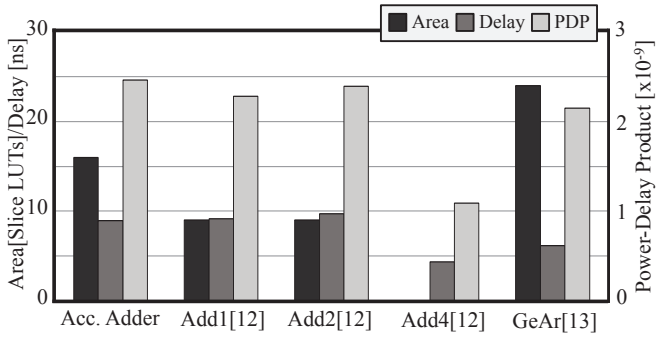

Fig. 1: FPGA Implementations of State-of-the-Art ASIC-based Approximate Adders [12], [13]

As can be observed in fig. 1, compared to the accurate version, the state-of-the-art ASIC-based approximate adders show asymmetric reductions in area and Power Delay Product (PDP), and an increase in critical path delay. However, when synthesized for ASIC-based systems, these approximate adders offer latency, area and power reductions of at least 41%, 46% and 74%, respectively, for bigger adder blocks, as per the studies shown in [16]. Proportional reductions in area, power and latency are not achieved as these approximations target transistor or gate-level truncations, leading to significant efficiency gains in ASICs but not in FPGAs. The most important factor in distinguishing ASICs and FPGAs is the way logic functions are

realized. The basic building blocks for generating the required logic, in case of ASICs, are the logic gates, whereas in case of FPGAs, they are the lookup tables (LUTs) made of SRAM elements. Therefore, the approximations techniques for FPGAs should be amenable to the LUT structures instead of aiming at reducing the logic gates.

B. Novel Contributions

We address the above challenges through our following novel contributions:

- **Generic Design Methodology:** We propose DeMAS, a generic methodology to design approximate adder architectures by analyzing the architectural features and resources of the target FPGA. This enables the reader to design multiple architectural versions of the approximate adder which could be used as per requirement, based on the platform and constraints. Furthermore, various approximate versions can be analyzed as trade-off points with diverse area, latency, energy, and output quality properties.
- **Approximate Adders for Xilinx 7-series FPGAs:** Using our novel methodology we were able to design 8 unique adder architectures which are implemented using various lookup table primitives, offered by Xilinx, employing different kinds of approximations based on the logic which can be compacted in these primitives. These adders have also been characterized in terms of area, latency, energy, avg. error magnitude, and avg. relative error.
- **Open-source Library:** We release the source codes (both hardware RTL in VHDL and behavioral models in MATLAB) of the approximate adders customized for the Xilinx 7-series FPGAs as an open-source library accessible on-line at https://sourceforge.net/projects/approxfpgas/. Moreover, we have also added all the scripts and constraint files to ensure reproducible results and fair comparison for future works by other researchers.

Paper Organization: Section II presents our generic design methodology, DeMAS, for designing the approximate adders, our proposed adder designs, the technique for building larger approximate adder blocks and an overview of our experimental setup. Section III presents the results, followed by conclusion in Section IV.

II. Our DeMAS Methodology for Approximate Computing in FPGAs

There are four key steps in our methodology to design approximate adders for FPGA-based systems. Fig. 2 presents an overview of DeMAS, our novel generic design methodology.

A. Extracting Architectural Features of Target FPGA Platform

As explained in section I-A, the reason why ASIC-based designs do not offer significant reductions is because they do not take into account the architectural differences between ASICs and FPGAs. Hence, as the first step, we need to analyze the FPGA type and its architectural features for which the approximate adders are being designed. In this paper, as an example, we target the Xilinx 7-series FPGA family of devices, namely, the Virtex-7 device 7VX330T.

The basic building block of an FPGA are the Configurable logic blocks or CLBs. They are used to implement any kind of logic function using the switching/routing matrix. Each CLB consists of two slices as shown in Fig. 2(a). Xilinx Virtex-7 family arranges all the CLBs in columns by deploying Advanced Silicon Modular Block (ASMBL) architecture. Each slice in this device consists of four 6-input LUTs, eight flip-flops and an efficient carry chain logic. The slices act as the main function generators of any FPGA and in the Virtex-7 family they are categorized as SLICEL or logic slices, and SLICEM or memory slices.

The lookup tables present in these slices are 5x2 LUTs. This LUT6_2 is fabricated using two LUT5s and a multiplexer as shown in fig. 2(b). These LUT5s are the basic SRAM elements which are used to realize the required logic function, by storing the output sequence of the truth-table in 1-bit memory locations, which are accessed using the address lines acting as inputs. These LUTs are made accessible using a wide range of lookup table primitives offered by the Xilinx UNISIM library [25], ranging from LUT1 to LUT6. These LUT primitives are instantiated with an INIT attribute, which is the truth

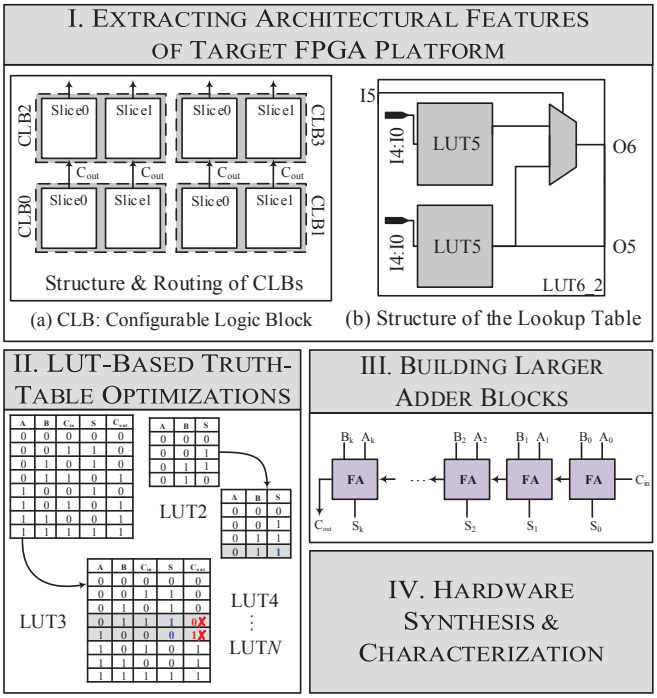

Fig. 2: Overview of DeMAS Showing Key Steps

table of the function required based on the input logic. The LUT primitives are used to implement the required logic function which are then compacted and mapped onto the fabric resources available on the FPGA. Each of these LUT primitives take in an equivalent number of 1-bit inputs, and produce a unique 1-bit output. However, at the hardware level, each of these primitives are physically mapped to one of the two LUT5s present in the four LUT6_2 fabrics in a given slice of the CLB. As per studies shown in [26], the use of LUT primitives allows for Xilinx to efficiently optimize the combining and mapping of LUT primitives to reduce the area and latency of the synthesized designs. We use these LUT primitives to achieve significant area and performance gains for the approximate adder designs.

B. LUT-Based Truth-table Optimizations

To compact the logic function into the truth table and reduce the critical path delay and area of the adders, we implement the simple carry chain truncation technique. We analyze the truth tables of 1- and 2-bit full adders, and try to simplify and reduce them to occupy lesser number of lookup tables. Table I presents the 1-bit FA truth-table.

TABLE I: Truth Table for the 1-bit Full Adder

Inputs			Accurate		Truncated		Optimized	
A_k	B_k	C_{in}	S_k	C_{out}	S_k	C_{out}	S_k	C_{out}
0	0	0	0	0 ✓	0	0 ✓	0	0 ✓
0	0	1	1	0 ✓	1	0 ✓	1	0 ✓
0	1	0	1	0 ✓	1	0 ✓	1	0 ✓
0	1	1	0	1 ✓	0	0 ✗	1	0 ✗
1	0	0	1	0 ✓	1	1 ✗	0	1 ✗
1	0	1	0	1 ✓	0	1 ✓	0	1 ✓
1	1	0	0	1 ✓	0	1 ✓	0	1 ✓
1	1	1	1	1 ✓	1	1 ✓	1	1 ✓

First, we eliminate the carry chain by equating the C_{in} of a given stage with one of the inputs (A, in our case) of the previous stage. This generates an error in 2 of the 8 possible cases, with a constant error magnitude of 2. However, the error magnitude can be further reduced to 1 by modifying the S_k bit for the error cases. Similarly,

we draw multiple 1-/2-bit half-/full- adder truth tables, approximate them by truncating the carry chain and equating C_{in} of the k^{th} stage with A_{k-1}, followed by modifying the sum bits to reduce the error magnitude.

For the above example, an LUT3 primitive, with inputs A, B and C_{in}, needs to be instantiated with the INIT value of '96', the hexadecimal equivalent of '10010110', to compute the sum-bit for the carry truncated adder design. INIT attribute is modified to '8E' for the error-optimized adder. These values are the data stored in 1-bit SRAM cells with inputs acting as the address lines to procure the required data. Using this methodology, we have designed 8 different approximate adders as shown in fig. 3. We discuss these designs in the following sub-sections.

1) Approximate Adder-1: This design utilizes a single LUT2 primitive to implement a half-adder circuit. The sum bit of a given stage depends solely on the two inputs of the same stage and the carry is neither computed nor taken into account when the computations are being performed as shown in Fig. 3(a). The LUT2 primitive is instantiated with the INIT value 'E'.

2) Approximate Adder-2: The truth table approximation shown in Table I is implemented in this design. As explained before, a single LUT3 primitive is used to implement an approximate FA circuit. This adder uses the simplest approximation technique in which the carry computation circuit is eliminated, and the sum-bit is modified to reduce the error magnitude. Fig. 3(b) depicts the adder design. S_0 is computed accurately using the inputs A_0, B_0, and C_{in}, whereas C_{out} is equated to the input A_0. The INIT value for the LUT3 is '8E'.

3) Approximate Adder-3: Adder-3 is a 2-bit adder circuit using LUT2 and LUT4 primitives that compute both S_0 and S_1 as depicted in Fig. 3(c). The carry is neither utilized nor computed. This architecture differs from the Adder-1 as the computation of every S^{th}_{n+1} bit is performed with a higher accuracy, with the intermediate carry-out from the previous stage taken into account. The INIT value for the LUT2 and LUT4 are 'E' and '80EC', respectively.

4) Approximate Adder-4: Approximate Adder-4 is also a 2-bit adder design implemented using LUT2 and LUT5 primitives. However, the S^{th}_{2n+1} bit's computational accuracy is further increased by accounting for the carry-in as shown in Fig. 3(d). The INIT value for the LUT2 and LUT5 are 'E' and 'E080FEF8', respectively.

5) Approximate Adder-5: This architecture, using LUT3 and LUT4 primitives, is designed as a 2-bit adder circuit. In the previous design, accuracy of S_1 was increased, whereas in this adder design the accuracy of S_0 is increased by using an LUT3 primitive instead of an LUT2 as shown in Fig. 3(e). The INIT value for the LUT3 and LUT4 are '8E' and '80EC', respectively.

6) Approximate Adder-6: This adder architecture utilizes the LUT3 and LUT5 primitives to efficiently exploit the LUT6_2 fabrics present in the slice. This adder's accuracy is the highest of all the approximate adder designs presented in this paper as the computational accuracy of both S_0 and S_1 are increased by accounting for the C_{in} as shown in Fig. 3(f). The INIT value for the LUT3 and LUT5 are '8E' and 'E080FEF8', respectively.

7) Approximate Adder-7: In this adder design, the LUT5 primitive is used to compute S_1 which is further equated to S_0 (i.e., $S_{2n} = S_{2n+1}$) as depicted in Fig. 3(g). A single LUT5 primitive is used in this design with C_{out} equated to A_1. The INIT value for LUT5 is 'E080FEF8'.

8) Approximate Adder-8: Like adder-7 architecture, a single LUT4 primitive is used to compute both S_1 and S_0 with $S_{2n} = S_{2n+1}$, as shown in Fig. 3(h). The INIT value for LUT4 is '80EC'.

C. Building Larger Adder Blocks

The designed half/full adder circuits can be extended to build larger 8-, 16-, 32-, or N-bit adders. Without loss of generality, for illustrative reasons, we use the carry adder design to implement and/or extend the smaller adder blocks. There are two techniques through which the adders can be approximated. Either the adder can be built solely using the designed approximate blocks (i.e., adders-1 to 8) or a mixture of accurate MSB and approximate LSB can be implemented, to avoid high error magnitude at the output. When implementing half adder circuits in the LSBs, the C_{in} for the next stage is equated to one of the inputs (A) of the current stage. For simplicity, we use a single type of approximate adder design instead of a combination of 2 or more types.

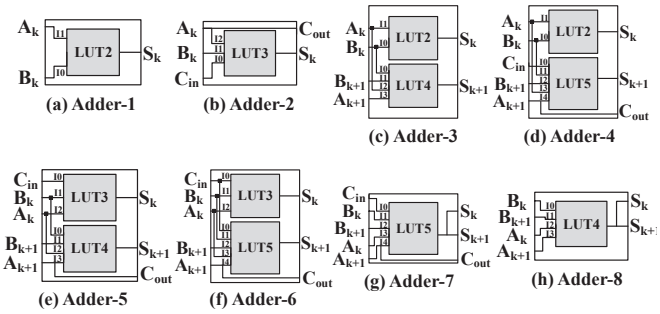

Fig. 3: Eight Different Designs for Approximate Adders Customized for Xilinx 7-series FPGAs

D. Hardware Synthesis & Characterization

The approximate adders obtained in the previous stage can then be categorized and analyzed for two different aspects, resource consumption and the quality of output. To obtain the resources consumed by the adder designs, we utilize the Xilinx ISE 14.7 design framework to synthesize and implement these designs on the Virtex-7 family device 7VX330T, i.e., our target platform. The maximum compression flag was enabled with the optimization goal set to latency, so as to reduce area and delay, simultaneously. They are characterized based on metrics like area (number of LUTs), latency, and energy consumption. An in-depth error analysis is performed using the MATLAB level behavioral models of the designed adders to understand the output quality of the adder designs, using metrics like average error magnitude and relative error. Fig. 4 illustrates our experimental setup and tool-flow for developing and analyzing the adder designs. Besides providing the results for our proposed approximate adders, we also provide the results for the existing state-of-the-art approximate adder designs to ensure a fair comparison.

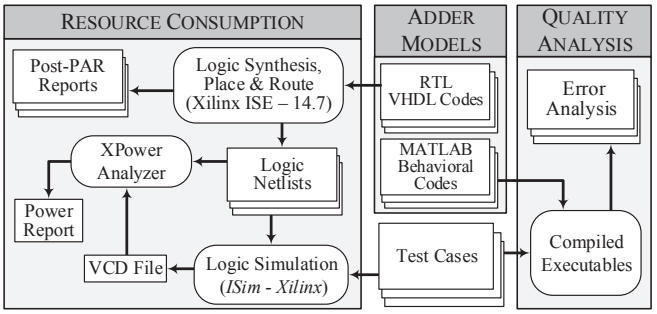

Fig. 4: Tool-flow and Experimental Setup

III. Results & Discussion

We simulated the proposed approximate adders along with the state-of-the-art ASIC-based designs and the FPGA-based approximate adder architecture presented in [27]. These designs were evaluated for a 16-bit system, with either 8 or 16 LSBs approximated. Fig. 5 presents the area, latency, power-delay product and quality of all the approximate adder designs.

Adders {B1, B4}, {B2, B5} and {B3, B6} represent the approximate adders presented in [12] with {8, 16} LSBs approximated. Note that the {B3, B6} adder design employs no logic, i.e., the outputs are directly connected to the inputs of the given stage, and hence require no logic or LUTs. Adder B7 is the LUT-based approximate adder design proposed in [27]. This technique truncates the carry chain in the middle, with two segments computing the output bits simultaneously, thereby reducing the latency of the design. Designs C1 – C8 depict the proposed approximate adders-1 to 8, with 8 LSBs approximated, whereas adders D1 – D8 employ approximation in all 16 LSBs.

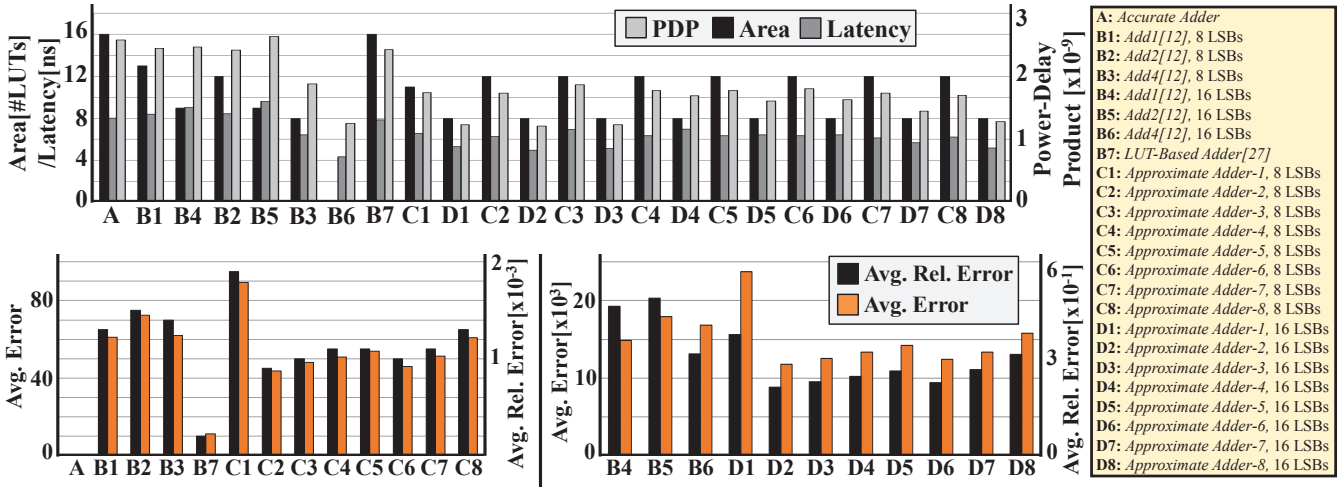

Fig. 5: Area, Latency, PDP and Quality of 16-bit Approximate Adders

Compared to the existing state-of-the-art approximate adders, which provide menial and/or asymmetrical reductions in PDP, latency, and area, our adders are successful in achieving area, latency and PDP reductions of up to 50%, 38% and 53%, respectively, when all the bits are approximated. Among all the adder designs presented, the adder B7 produces the least amount of average or relative error. It is, however, not successful in achieving any significant reductions in area, latency and PDP. On the other hand, the proposed adder-1 produces the highest average error due to the type of approximation employed. The proposed approximate adders cover a wide range of the design space, with adders like C2 – C7 outperforming the ASIC-based designs, in terms of accuracy and quality of output, as well as reducing the resources consumed.

The number of lookup tables occupied by all the proposed adder designs are the same, as the LUT primitives are mapped to the same LUT6_2 fabric available on the FPGA. However, the use of LUT primitives enables the user to exploit the underlying Xilinx optimization mechanisms to reduce the delay of the adder designs. As can be observed in fig. 5, the use of underlying LUT primitives offers different delays for the eight adder designs proposed in this paper, similar to the findings presented in [26]. As expected, Xilinx optimizes the latency and area by combining and mapping the LUT primitives, this is most prevalent in the proposed adder-1, which when configured implicitly leads to sum bits $\{S_0, S_1\}$ being computed in one LUT, $\{S_2, S_3\}$ being computed in another, and so on. Whereas when using LUT primitives like LUT-2, the Xilinx tool-flow optimizes, combines and maps different sum bits like $\{S_0, S_{11}\}$ to be computed in a single LUT, thereby reducing the delay of the adder. The adder designs D7 and D8 are quite similar, of which the former is implemented using LUT5 and the latter using LUT4 primitives. However, the design D7 has a higher critical path delay as compared to the D8 adder design. This difference in delay is observed because of the truncated inputs, and the optimal mapping of the LUT primitives by Xilinx, which changes the critical path and the delay of the system.

IV. Conclusion

In this paper, we have proposed a novel generic methodology to design approximate adder components based on the target FPGA's system resources. Using this methodology we have designed eight novel 1-/2-bit approximate adder architectures for the Xilinx 7-series FPGAs. These adders are successful in achieving reductions of 50%, 38% and 53%, in the area, latency and PDP, respectively. Most of the errors exhibited by these approximate adders have comparatively low value concentration and can replace accurate adders for a wide-range of error-resilient applications. Our work is open source and the approximate adder library, along with behavioral models, is accessible on-line at https://sourceforge.net/projects/approxfpgas/. We further plan to explore the reconfigurable nature of FPGAs to exploit approximate architectures and their utilization in higher order accelerators to achieve energy-efficient applications.

References

[1] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *ACM/IEEE DAC*, 2013.

[2] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, "Invited: Cross-layer approximate computing: From logic to architectures," in *ACM/IEEE DAC*, 2016.

[3] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys*, 2016.

[4] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design Test*, 2016.

[5] M. Imani, A. Rahimi, and T. S. Rosing, "Resistive configurable associative memory for approximate computing," in *DATE*. ACM/IEEE, 2016.

[6] K. V. Palem, L. N. Chakrapani, Z. M. Kedem, A. Lingamneni, and K. K. Muntimadugu, "Sustaining moore's law in embedded computing through probabilistic and approximate design: Retrospects and prospects," in *CASES*. ACM, 2009.

[7] D. Shin and S. K. Gupta, "Approximate logic synthesis for error tolerant applications," in *ACM/IEEE DATE*, 2010.

[8] S. Misailovic, M. Carbin, S. Achour, Z. Qi, and M. C. Rinard, "Chisel: Reliability- and accuracy-aware optimization of approximate computational kernels," *SIGPLAN Not.*, 2014.

[9] W. Baek and T. M. Chilimbi, "Green: A framework for supporting energy-conscious programming using controlled approximation," *SIGPLAN Not.*, 2010.

[10] D. Mahajan, K. Ramkrishnan, R. Jariwala, A. Yazdanbakhsh, J. Park, B. Thwaites, A. Nagendrakumar, A. Rahimi, H. Esmaeilzadeh, and K. Bazargan, "Axilog: Abstractions for approximate hardware design and reuse," *IEEE Micro*, 2015.

[11] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," *SIGPLAN Not.*, 2012.

[12] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE TCAD*, 2013.

[13] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *ACM/IEEE DAC*, 2015.

[14] M. A. Hanif, R. Hafiz, O. Hasan, and M. Shafique, "Quad: Design and analysis of quality-area optimal low-latency approximate adders," in *DAC*. ACM, 2017.

[15] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *2011 24th International Conference on VLSI Design*, 2011.

[16] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, and J. Henkel, "Architectural-space exploration of approximate multipliers," in *ACM ICCAD*, 2016.

[17] S. Mazahir, O. Hasan, R. Hafiz, M. Shafique, and J. Henkel, "Probabilistic error modeling for approximate adders," *IEEE TC*, vol. 66, 2017.

[18] S. Mazahir, O. Hasan, R. Hafiz, and M. Shafique, "Probabilistic error analysis of approximate recursive multipliers," *IEEE TC*, vol. 66, 2017.

[19] W. El-Harouni, S. Rehman, B. S. Prabakaran, A. Kumar, R. Hafiz, and M. Shafique, "Embracing approximate computing for energy-efficient motion estimation in high efficiency video coding," in *DATE*, 2017.

[20] A. K. Mishra, R. Barik, and S. Paul, "iACT: A software-hardware framework for understanding the scope of approximate computing," in *WACAS*, 2014.

[21] R. Nair, "Big data needs approximate computing: technical perspective," *Communications of the ACM*, 2015.

[22] J. Bornholt, T. Mytkowicz, and K. S. McKinley, "Uncertain<T>: Abstractions for uncertain hardware and software," *IEEE Micro*, 2015.

[23] Open-source low-power approximate computing library. [Online]. Available: https://sourceforge.net/projects/lpaclib/

[24] Open-source approximate adder library. [Online]. Available: http://sourceforge.net/projects/approxadderlib/

[25] Xilinx. (2013) Xilinx 7 Series FPGA Programmable Guide for HDL Designs. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/7series_hdl.pdf

[26] A. Ehliar, "Optimizing xilinx designs through primitive instantiation," in *7th FPGAworld Conference*. ACM, 2010.

[27] A. Becher, J. Echavarria, D. Ziener, S. Wildermann, and J. Teich, "A lut-based approximate adder," in *FCCM*. IEEE, 2016.