# Area-Optimized Low-Latency Approximate Multipliers for FPGA-based Hardware Accelerators

Salim Ullah[1], Semeen Rehman[2], Bharath Srinivas Prabakaran[2], Florian Kriebel[2], Muhammad Abdullah Hanif[2], Muhammad Shafique[2], Akash Kumar[1]

[1]Technische Universität Dresden, Germany

[2]Vienna University of Technology, Austria

Corresponding Author's Email: salim.ullah@tu-dresden.de

## ABSTRACT

The architectural differences between ASICs and FPGAs limit the effective performance gains achievable by the application of ASIC-based approximation principles for FPGA-based reconfigurable computing systems. This paper presents a novel approximate multiplier architecture customized towards the FPGA-based fabrics, an efficient design methodology, and an open-source library. Our designs provide higher area, latency and energy gains along with better output accuracy than those offered by the state-of-the-art ASIC-based approximate multipliers. Moreover, compared to the multiplier IP offered by the Xilinx Vivado, our proposed design achieves up to 30%, 53%, and 67% gains in terms of area, latency, and energy, respectively, while incurring an insignificant accuracy loss (on average, below 1% average relative error). Our library of approximate multipliers is open-source and available online at https://cfaed.tu-dresden.de/pd-downloads to fuel further research and development in this area, and thereby enabling a new research direction for the FPGA community.

## 1 INTRODUCTION AND RELATED WORK

Multiplication is one of the basic arithmetic operations, used extensively in the domain of digital signal and image processing. FPGA vendors, such as Xilinx and Intel, provide DSP blocks to achieve fast multipliers. Despite the high performance offered by the DSP blocks, their usage might not be efficient in terms of overall performance and area requirements for some applications. Table 1 compares two different implementations of Reed-Solomon and JPEG encoders[1] for Virtex-7 series FPGA (7VX330T device) using Xilinx Vivado 17.1. The routing delay, caused by the location of the allocated DSP blocks, has resulted in higher latency for DSP-based implementation of Reed-Solomon encoder. For small applications, it may be possible to perform manual Floorplanning to optimize the overall performance of an application, but for complex applications having contending requirements for FPGA resources, it may not be possible to optimize the placement of required FPGA resources for enhancing the performance gains. Similarly, the implementation of the JPEG-encoder shows a large number of DSP blocks (56% of the total available DSP blocks) utilization. Such applications can exhaust the available DSP blocks for other performance critical operations. As a result, other applications executing concurrently on the same FPGA will opt for using LUT-based multipliers.

[1]Source codes from *http://opencores.org/projects*



*DAC '18, June 24–29, 2018, San Francisco, CA, USA*



ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3195996

Similar results about DSP blocks utilization and overall application performance are also reported by [13]. In short, despite the availability of DSP blocks, the need of LUT-based multipliers is inevitable. That is why Xilinx and Intel also provide logic-based soft multipliers [10, 13, 20]. Multiplier designs like [12] and [18] have also considered the efficient utilization of FPGA resources for providing high performance. However, a wide range of applications do not require accurate intermediate computations and their operations can be approximated to further improve performance and energy efficiency. These applications have inherent resilience to approximation induced errors and thereby demonstrate the ability to produce viable outputs despite some of the input-data/intermediate computation being incorrect or approximate. Examples of such applications can be found in the domains of image/signal processing, machine learning and various other probabilistic algorithms [3].

**Table 1: Comparison of logic vs DSP blocks based implementations**

| Design | DSP Blocks Enabled | | | DSP Blocks Disabled | | |
|---|---|---|---|---|---|---|
| | Critical Path Delay [ns] | Total No. of LUTs | Total No. of DSP Blocks | Critical Path Delay [ns] | Total No. of LUTs | Total No. of DSP Blocks |
| Reed-Solomon Encoder | 5.115 | 2826 | 22 | 4.358 | 2867 | 0 |
| JPEG Encoder | 8.637 | 71362 | 631 | 9.732 | 14780 | 0 |

Using the principles of *approximate computing*, works in [4, 5, 8, 11] and [1, 6, 7, 14–16, 19] suggest the use of functional approximations for designing different types of approximate adders and multipliers with different performance gains. However, because of the inherent architectural differences between FPGAs and ASICs, most of these techniques provide limited or no performance gains when directly synthesized for the FPGA-based systems. To further emphasize the need for designing FPGA-based approximate modules, we present the following motivational case study.

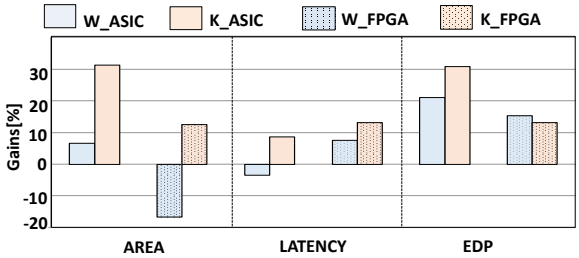



**Figure 1: Cross-platfrom comparison of area, latency and EDP**

## 1.1 Motivational Case Study

Fig. 1 compares the ASIC-based area, latency and *Energy-Delay-Product (EDP)* gains of two state-of-the-art approximate multipliers, *"W"*, presented in [19], and *"K"*, described in [6], with their FPGA-based implementations. The gains for ASIC-based implementations have been obtained from [19] and [6], whereas for the FPGA-based implementations, Xilinx Vivado 17.1 tool for the 7VX330T device of Virtex-7 family has been used. As shown by the analysis results, the gains offered by the ASIC-based implementation are not proportionally translated to FPGA-based implementation. The area and EDP gains offered by *W* and *K* are reduced for FPGA-based implementation. However, the latency gains have improved for both multipliers. This lack of similar performance gains for FPGA-based systems is the result of the architectural differences between ASICs and FPGAs. In ASIC-based designs, logic gates are deployed for the implementations of different logic circuits, thus a full control over resource utilization at a fine granularity is possible. However, FPGA-based computational blocks are composed of entirely different entities, i.e., look-up tables (LUTs) where configuration bits are used to implement a certain circuit. This poses a **research challenge** of defining LUTs-based approximations for FPGA-based systems to achieve significant performance gains.

## 1.2 Our Novel Contributions

To address the above research challenge, this paper presents a novel approximate multiplier architecture, that has been specifically designed for FPGA-based systems. The proposed method utilizes LUTs for the generation of approximate partial products. As most of the modern LUTs have six inputs, therefore, to completely utilize a LUT, this paper presents a novel approximate 4×2 multiplier as an elementary module. In order to reduce the number of output errors, we then perform different FPGA-specific optimizations and generate an approximate and asymmetric 4×4 multiplier. It has increased output accuracy and reduced latency and area requirements as compared to the state-of-the-art approximate multipliers. To the best of our knowledge, this work is the first attempt towards FPGA-specific approximate multipliers by utilizing LUTs and associated carry chains to generate approximate partial products. To further explore the efficacy of the proposed 4×2 and 4×4 elementary multipliers, this paper also presents the approximate addition of the generated approximate partial products.

Our approximate multipliers have been characterized using the following quality metrics (as also adopted by the literature [1, 6, 9]).

- Number of Error Occurrences
- Maximum Error Magnitude
- Average Relative Error
- Number of Maximum Error Case Occurrences

The proposed asymmetric 4×4 multiplier has total 6 error cases with fixed error magnitude for a uniform input distribution. For different real-world applications with non-uniform input data sets, the asymmetric nature of the proposed multiplier can be utilized for improving the output accuracies. This is also verified by our experimental analysis in section 5, where the mutual swapping of the input values to the multiplier results in improving the final output accuracies.

The rest of the paper is organized as follows: Section 2 presents the preliminaries and the inspiration for designing approximate 4×2 multiplier as the basic block for designing higher order multipliers, Section 3 describes our novel design of 4×2 and 4×4 multipliers, followed by the description of designing higher order multipliers using approximate sub-components in Section 4. Finally, Section 5 describes the implementation and analysis results of our approximate multipliers.

# 2 PRELIMINARIES

The proposed design has been implemented using Xilinx FPGAs, however, the presented methodology can also be implemented on FPGAs from other vendors, such as Intel which provides fracturable 6-input LUTs and carry chains.

A slice in the configurable logic block (CLB) of Xilinx 7-series

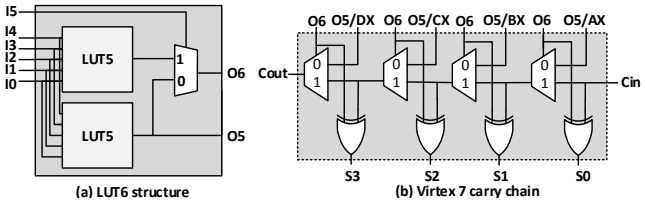



**Figure 2: Xilinx FPGA slice structure [21]**

FPGAs have four 6-input LUTs (commonly referred as *LUT6_2*) along with eight flip-flops for registering LUTs outputs and a single 4-bit long carry chain [21]. A *LUT6_2* can be used to implement either a single 6-bit combinational function, using O6 output bit, or two 5-bit combinational functions, using O5 and O6 output bits, by defining an INIT value which describes all the possible input combinations for which a logic value "1" is required at the output. For example, an INIT value of *0000000000000002(hex)* for *LUT6_2* defines to produce outputs *O5 = 1 & O6 = 0* for input combination *100001*. Besides the implementation of combinational functions, these 6-input LUTs are also used for controlling the associated carry chain; as shown in Fig. 2(b). The carry chain implements a 4-bit carry-look ahead adder using O5 as carry-generate signal and O6 as carry-propagate signal.

A Performance/Area Optimized Elementary Multiplier Module, targeted for FPGAs, should efficiently utilize the available *LUT6_2* and associated carry chains in FPGAs. The $2 \times 2$ multipliers, as used by [19] and [6], under-utilize *LUT6_2* and therefore has been excluded from the list of potential elementary multipliers. The only two potential multiplier designs, which utilize all the inputs of a *LUT6_2*, are $3 \times 3$ and $4 \times 2$ multipliers. However, a $3 \times 3$ multiplier is not a feasible option for the implementation of higher order multipliers, e.g. $4 \times 4$ and $8 \times 8$ multipliers. A $4 \times 4$ multiplier requires one $3 \times 3$, one $1 \times 4$ and one $3 \times 1$ multipliers [2]. This limited applicability of a $3 \times 3$ multiplier results in filtering it out from our selection of an elementary multiplier module. The only feasible elementary design is a $4 \times 2$ multiplier, which thoroughly utilizes lookup tables of state-of-the-art FPGAs. A $4 \times 4$ multiplier can be implemented using two instances of a $4 \times 2$ multiplier. This paper uses $4 \times 2$ multiplier as elementary block for designing higher order multipliers. Using $4 \times 2$ multipliers, a $4 \times 4$ multiplier with improved output accuracy has been presented.

# 3 APPROXIMATE DESIGN OF ELEMENTARY MULTIPLIER MODULES

Before presenting the approximate $4 \times 4$ multiplier design, we present the approximate $4 \times 2$ multiplier design in the next subsection.

## 3.1 Approximate Design of 4 × 2 Multiplier

An accurate $4 \times 2$ multiplier generates a 6-bit output with the following optimized logic equations for A($A_3A_2A_1A_0$) and B($B_1B_0$) as multiplicand and multiplier respectively:

$$P_0 = B_0A_0 \quad (1)$$

$$P_1 = B_1{}'B_0A_1 + B_1B_0{}'A_0 + B_1A_1{}'A_0 + B_0A_1A_0{}' \quad (2)$$

$$P_2 = B_1{}'B_0A_2 + B_1B_0{}'A_1 + B_0A_2A_1{}' + B_1A_2{}'A_1A_0{}' + B_1A_2A_1A_0 \quad (3)$$

$$P_3 = B_1{}'B_0A_3 + B_1B_0{}'A_2 + B_1A_3{}'A_2A_1{}' + B_0A_3A_2{}'A_1{}' + B_1B_0A_3{}'A_2{}'A_1A_0 + B_0A_3A_2A_1 + B_0A_3A_1A_0{}' \quad (4)$$

$$P_4 = B_1B_0{}'A_3 + B_1A_3A_2{}'A_1{}' + B_1A_3A_2{}'A_0{}' + B_1B_0A_3{}'A_2A_1 \quad (5)$$

$$P_5 = B_1B_0A_3A_2 + B_1B_0A_3A_1A_0 \quad (6)$$

As $P_0$, $P_1$ and $P_2$ each depend on less than six shared variables i.e. $A_0$, $A_1$, $A_2$, $B_0$ and $B_1$, therefore, any two of these three least significant product bits can be generated using a single LUT6_2. The remaining four product bits will require four separate LUTs for implementation. An area and energy efficient approximation is to accommodate the six product bits in four LUTs i.e. a single slice. Truncation of $P_0$ limits the output error to the least significant product bit and the final output accuracy to 75% with maximum error magnitude of "1" for all input combinations. Approximation of any other product bit results in a higher magnitude of error in the final output. The proposed approximate design of 4 × 2 multiplier uses 4 LUTs for its implementation by truncating "$P_0''$" and generating "$P_1''$" and "$P_2''$" by a single LUT6_2.

## 3.2 Approximate Design of 4 × 4 Multiplier

The approximate design of 4 × 4 multiplier requires two 4 × 2 multipliers, consuming eight LUTs for partial products generation. For multiplicand A($A_3A_2A_1A_0$) and multiplier B($B_3B_2B_1B_0$), the first 4×2 multiplier takes A($A_3A_2A_1A_0$) & $B_0$($B_1B_0$) and the second 4×2 multiplier occupies A($A_3A_2A_1A_0$) & $B_1$($B_3B_2$) as input operands.

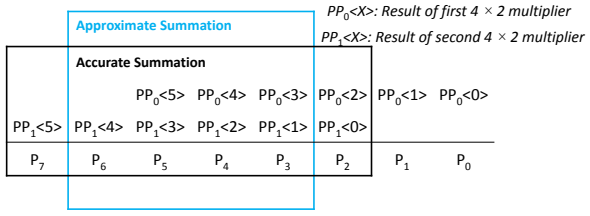



**Figure 3: 4×4 using 4×2 multipliers**

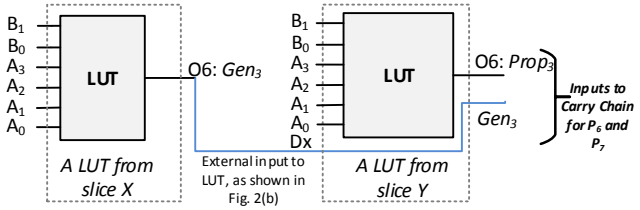



**Figure 4: Implementation of $Gen_3$ and $Prop_3$ for $P_6$ and $P_7$**

As shown by the black box in Fig. 3, the accurate summation of the approximate partial products generated by the two 4 × 2 multipliers requires the use of two carry chains. Therefore, the approximate 4 × 4 multiplier, with accurate summation of partial products, requires 16 LUTs (2 LUTs wasted by the second carry chain). Due to the truncation of $PP_0 < 0 >$ and $PP_1 < 0 >$ in Fig. 3, this 4 × 4 multiplier implementation has an average relative error of 0.049 with an error probability of 0.375 for a uniform input distribution. However, the proposed design performs approximate addition along with FPGA-specific optimizations of second 4 × 2 multiplier and uses one single carry chain for partial products summation, as shown by the blue rectangle in Fig. 3. Our optimizations not only provides area gains but also significantly improves the total number of error cases by having only 6 erroneous outputs. Our proposed optimization uses three LUT6_2s for the implementation of required *Carry Propagate* and *Carry Generate* signals to compute $P_3$, $P_4$ and $P_5$ product bits. Since $PP_1$<4> and $PP_1$<5> share same six operands, therefore our design does not compute $PP_1$<4> and $PP_1$<5> explicitly for subsequent addition by the carry chain. The proposed approach, as shown in Fig. 4, computes the respective *Carry Propagate 'Prop$_3$'* and *Carry Generate 'Gen$_3$'* signals for the computation of $P_6$ and $P_7$ directly from the multiplier and multiplicand bits by implicitly generating $PP_1$<4> and $PP_1$<5>. This implicit implementation of $PP_1$<4> and $PP_1$<5> saves one LUT as compared to their explicit computation. In order to improve the output accuracy, the recovered LUT is then assigned for the accurate realization of $P_0$ and $P_2$. Since the computation of $P_3$ is also dependent on the carry-out from $P_2$, therefore, the corresponding LUT for $P_3$ besides, using $PP_0$<3> and $PP_1$<1> also utilize $A_0$, $B_2$ and $PP_0$<2> to resolve the effect of the missing carry-out from $P_2$. As carry propagate and carry Generate signals cannot be "1" simultaneously, all the cases where $A_0$, $B_2$, $PP_0$<2>, $PP_0$<3> and $PP_1$<1> are all "1" concurrently, will generate an error. In order to limit the error occurrences to a single product bit, $P_3$, we propose to correctly compute the carry Generate signal only. This decision limits the error to $P_3$ only with a fixed error magnitude of "8".

**Table 2: 4×4 multiplier error values**

| Multiplier | Multiplicand | Actual Product | Computed Result | Difference |
|---|---|---|---|---|
| 5 | 15 | 75 | 67 | 8 |
| 6 | 7 | 42 | 34 | 8 |
| 6 | 15 | 90 | 82 | 8 |
| 7 | 15 | 105 | 97 | 8 |
| 13 | 13 | 169 | 161 | 8 |
| 15 | 5 | 75 | 67 | 8 |

The proposed design has been implemented on 7VX330T device of Virtex-7 series FPGA and Tables 2 and 3 present the input operands with erroneous outputs and INIT values employed by each LUT along with input/output pins configuration respectively. However, the availability of 6-input LUTs and dedicated adders in other FPGA architectures, such as Intel Stratix 10, also make the proposed design portable to other architectures to obtain comparable performance gains. It is noteworthy that depending upon an application's input data, the proposed 4×4 multiplier may produce better result due to its asymmetric nature and the values presented in Table 2 only show the maximum number of possible error occurrences for uniform distribution of all input cases. Our proposed multiplier does not generate erroneous outputs for highlighted inputs, in Table 2, with multiplier and multiplicand mutually swapped. For achieving better output quality results, the proposed approach suggests an initial analysis of input data, before multiplication, to decide operands for multiplier and multiplicand. The asymmetric nature of the proposed multiplier and the analysis of input data for achieving better output accuracy are further explored in section 5.

# 4 DESIGNING HIGHER ORDER APPROXIMATE MULTIPLIERS

The proposed methodology utilizes the recursive approach of adding approximate 4×2 and 4×4 multipliers for implementing higher order multipliers as shown in Fig. 5(a). For the process of addition, the

**Table 3: LUTs' inputs and outputs pins configuration for approximate 4×4 Multiplier**

| LUT | LUT Input Pins Configuration | | | | | | INIT value (Hex) | LUT Output Pins Configuration | |
|---|---|---|---|---|---|---|---|---|---|
| | I5 | I4 | I3 | I2 | I1 | I0 | | O6 | O5 |
| $LUT_0$ | 1 | $B_1$ | $B_0$ | $A_2$ | $A_1$ | $A_0$ | *B4CCF00066AACC00* | $PP_0$<2> | $PP_0$<1> = $P_1$ |
| $LUT_1$ | $B_1$ | $B_0$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ | *C738F0F0FF000000* | $PP_0$<3> | |
| $LUT_2$ | $B_1$ | $B_0$ | $A_1$ | $A_2$ | $A_1$ | $A_0$ | *07C0FF0000000000* | $PP_0$<4> | |
| $LUT_3$ | $B_1$ | $B_0$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ | *F800000000000000* | $PP_0$<5> | |
| $LUT_4$ | 1 | $B_3$ | $B_2$ | $A_2$ | $A_1$ | $A_0$ | *B4CCF00066AACC00* | $PP_1$<2> | $PP_1$<1> |
| $LUT_5$ | $B_3$ | $B_2$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ | *C738F0F0FF000000* | $PP_1$<3> | |
| $LUT_6$ | $B_3$ | $B_2$ | $A_1$ | $A_2$ | $A_1$ | $A_0$ | *F800000000000000* | $Gen_3$ | |
| $LUT_7$ | 1 | 1 | $PP_0$<2> | $B_2$ | $B_0$ | $A_0$ | *5FA05FA088888888* | $P_2$ | $P_0$ |
| $LUT_8$ | 1 | $PP_1$<1> | $PP_0$<3> | $B_2$ | $A_0$ | $PP_0$<2> | *007F7F80FF808000* | $Prop_0$ | $Gen_0$ |
| $LUT_9$ | 1 | 1 | 1 | 1 | $PP_1$<2> | $PP_0$<4> | *6666666688888880* | $Prop_1$ | $Gen_1$ |
| $LUT_{10}$ | 1 | 1 | 1 | 1 | $PP_1$<3> | $PP_0$<5> | *6666666688888880* | $Prop_2$ | $Gen_2$ |
| $LUT_{11}$ | $B_3$ | $B_2$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ | *07C0FF0000000000* | $Prop_3$ | |

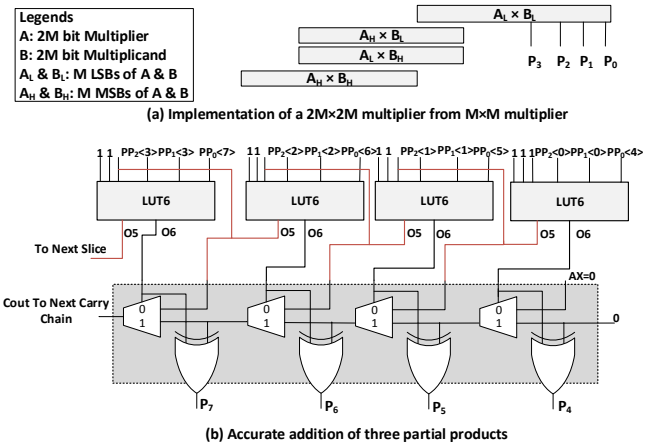



**Figure 5: Designing higher order multipliers from lower order multipliers: (a) Implementation of 2M×2M multiplier using M×M multipliers. (b) Generation of product bits $P_4$, $P_5$, $P_6$ and $P_7$ using carry chain**

proposed methodology utilizes accurate and approximate addition. The approximate multiplier *Ca* in Fig. 5(b), performs an accurate summation of the approximate partial products by using the associated carry chain. As shown in Fig. 5(b), $PP_0 < 4 > - PP_0 < 7 >$ from $A_L \times B_L$, $PP_1 < 0 > - PP_1 < 3 >$ from $A_H \times B_L$ and $PP_2 < 0 > - PP_2 < 3 >$ from $A_L \times B_H$ are added in one single step to produce final product bits $P_4 - P_7$ for an 8×8 multiplier. The O5 output of fourth LUT6 and Cout of carry chain in Fig. 5(b) are routed to next slice for generation of higher order product bits. The same process can be repeated for the implementation of arbitrary sizes of higher order multipliers. In the next section, we use approximate addition of the approximate partial products to obtain approximate multiplier Cc. An example for 8 × 8 multiplier is presented, but the same methodology can be followed to design arbitrary size of multiplier.

## 4.1 8 × 8 Approximate Multiplier Cc: Approximate Summation of Partial Products

For further improving the performance of the 8 × 8 multiplier, a *highly-inaccurate* approximation has been applied on the partial products summation, as shown by the blue dotted boxes in Fig. 6. All partial products are added using LUTs but without using carry-out from the preceding bit locations. The four least and most significant product bits are obtained without using addition, as shown in Fig. 6. The result of this highly-inaccurate approximate addition also signifies the high output accuracy of our basic 4×4 approximate multiplier. By utilizing sophisticated approximate addition, higher order approximate multipliers with higher output accuracies and area gains than those achievable with Cc can be obtained.
To characterize the proposed multipliers, a detailed analysis in terms of maximum error magnitude, number of error occurrences, number of maximum error occurrences, average error, area requirements, worst case latency, EDP requirements, output visual quality and peak signal to noise ratio (PSNR) values of approximate multipliers is presented in the next section.

# 5 RESULTS & DISCUSSION

## 5.1 Experimental Setup and Tool Flow

All presented multipliers have been implemented in VHDL and synthesized for 7VX330T device of Virtex-7 family using Xilinx Vivado 17.1. For EDP calculations, Vivado Simulator and Power Analyzers have been used. EDP and output accuracies of all proposed multipliers have been calculated for a uniform distribution of all input combinations. We compare the proposed multipliers for performance gains and output accuracies with W [19], K[6], library of 8-bit approximate multipliers EvoApprox8b[17], precision reduced 4×4 and 8×8 multipliers with three and four LSBs rounded to zero respectively, and Xilinx accurate multiplier IP[20].

The designed multipliers have also been implemented for the image smoothing accelerator of the SUSAN application to record the area savings offered by our novel approximate multipliers.

## 5.2 Evaluation and Characterization of Designed Multipliers

Table 4 presents the implementation results of our proposed approximate multipliers. For approximate 8×8 and 16×16 multipliers *Ca and Cc*, all sub-multipliers are approximate. *Cc* trades the output accuracy to provide area and latency gains. Table 5 presents an error analysis of our designed approximate multipliers in comparison with the state-of-the-art approximate multipliers and precision reduced 8×8 multiplier with four LSBs rounded to zero. The proposed multiplier *Ca* outperforms the existing approximate multipliers in terms of maximum error magnitude, average error, error occurrences and maximum error occurrences. The approximate multiplier *Cc* has higher maximum error magnitude than state-of-the-art W[19], however, the maximum error occurs only once for *Cc* while it occurs 31 times for W[19]. The precision reduced Mult(8,4) has highest number of maximum error occurrences. Regardless of its low average relative error, its high resource utilization, 350 LUTs,

**Table 4: Area and latency results of proposed multipliers**

| Multiplier Size | Area [LUTs] | Latency (ns) | Area [LUTs] | Latency (ns) |
|---|---|---|---|---|
| | Ca | | Cc | |
| 4×4 | 12 | 5.846 | 12 | 5.846 |
| 8×8 | 57 | 7.746 | 56 | 6.946 |
| 16×16 | 245 | 10.765 | 240 | 7.613 |

**Table 5: Error analysis of 8×8 approximate multipliers**

| Error Description | Approximate Architectures | | | | |
|---|---|---|---|---|---|
| | Ca | Cc | W[19] | K[6] | Mult(8,4) |
| Maximum Error Magnitude | 2312 | 8288 | 7225 | 14450 | 15 |
| Average Error | 54.1875 | 1592.265 | 1354.687 | 903.125 | 6.5 |
| Average Relative Error | 0.002917 | 0.129390 | 0.1438777 | 0.032549 | 0.0037 |
| Error Occurrences | 5482 | 52731 | 53375 | 30625 | 53248 |
| Maximum Error Occurrences | 14 | 1 | 31 | 1 | 2048 |

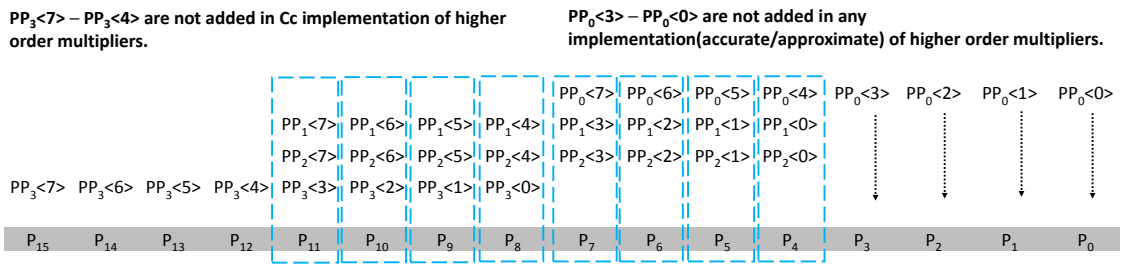



**Figure 6: 8×8 approximate multiplier and its approximate summation**

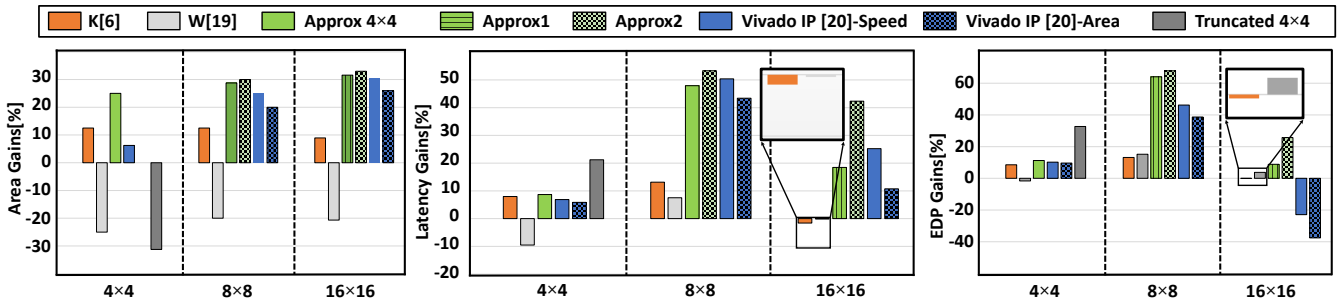
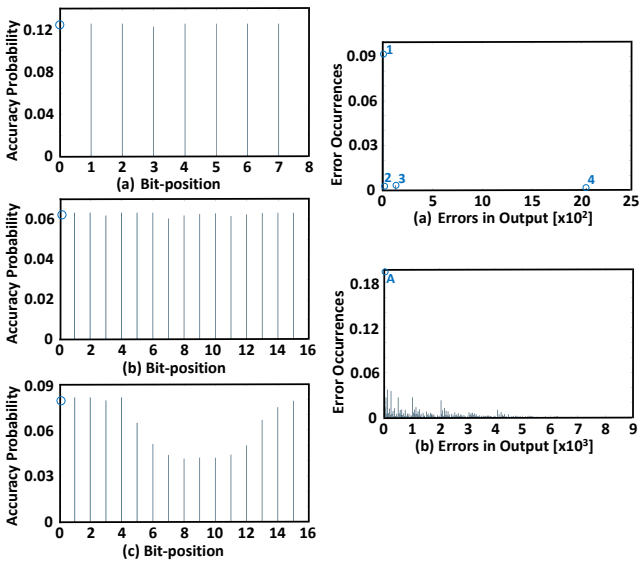



**Figure 7: Area, Latency, & EDP of 4, 8 and 16-bits Approximate Multipliers**




(a) Bit histograms of 4×4 and 8×8 Ca and Cc Multipliers (b) PMFs of 8×8 Ca and Cc Multipliers

**Figure 8: Probability of error in individual product bits**

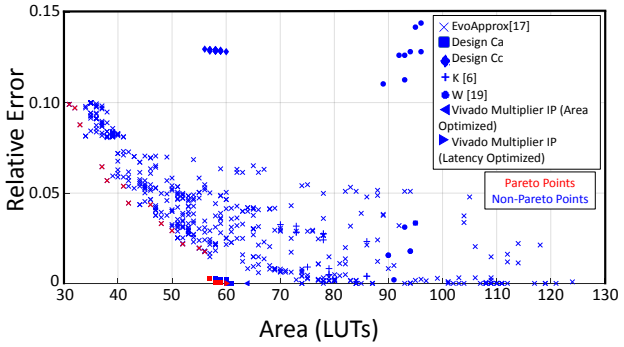



**Figure 9: Pareto optimal analysis of the proposed 8×8 multipliers with state-of-the-art approximate multipliers**

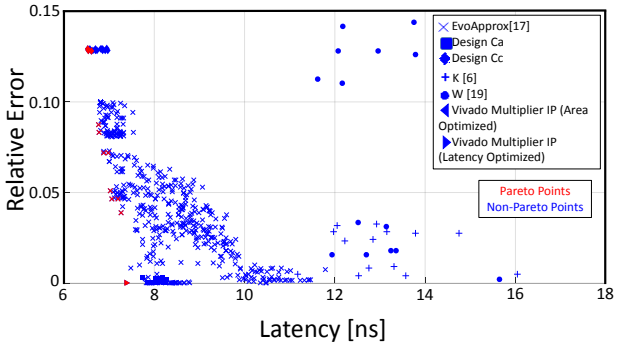



**Figure 10: Pareto optimal analysis of the proposed 8×8 multipliers with state-of-the-art approximate multipliers**

filters it out in Pareto analysis. To explore the erroneous bit values with their effect on final output and the frequency of error occurrences, Fig. 8 represents the normalized bit accuracy histograms and the normalized number of unique error occurrences for proposed multipliers. Our novel design restricts the errors to limited bits only. Except *Cc* multiplier, all other multipliers have few distinct errors. The low probability of getting accurate bit values for *Cc* is due to the highly-inaccurate approximate addition of the partial products. Such type of architectures, with limited distinct errors, can be easily configured to have an error-correction circuitry that can be turned on/off according to applications' requirements.

Besides enhanced output accuracies, the proposed multipliers are also better than state-of-the-art approximate multipliers *W*[19], *K*[6], Vivado's multiplier IP [20] (optimized for speed and area) and 4×4 truncated multiplier (3 LSBs have been truncated) in terms of

area, latency and EDP gains as shown in Fig. 7. These results have been normalized with respect to the area, latency and EDP results of Vivado's default accurate multiplier implementation. Our presented multipliers offer 25% — 31.5% area reduction, 8.6% — 53.2% reduction in latency and 8.86% — 67% gains in EDP when compared to the accurate multiplier implementation offered by Vivado.

Finally, Fig. 9 and Fig. 10 compare all possible configurations of the presented 8×8 multipliers and state-of-the-art multipliers W[19], K[6], EvoApprox8b[17] and area/latency optimized Xilinx multiplier IP[20] with respect to average relative error, occupied LUTs and critical path delay. The Pareto optimal analysis reveals that the number of non-dominated points reported by Evoapprox8b in [17] has significantly reduced for FPGA-based implementation. This analysis is in accordance with our observation of ASIC-based approximations less effective in producing comparable results for FPGA-based systems. The design points with very low average relative error and low area requirements are only provided by our proposed methodology. Similarly, our methodology offers design points with low critical path delay and low average relative error.

The proposed multipliers are also tested for the SUSAN application based image smoothing accelerator to observe area gains. Our approximations produced 17%, and 17.2% area gains for *Ca and Cc* multipliers respectively with insignificant output quality loss. Fig. 11 and Table 6 contrast the output visual qualities and the PSNR values of SUSAN image smoothing accelerator, using proposed approximate multipliers, accurate multiplier and state-of-the-art multipliers *W*[19] and *K* [6] respectively. Results show that our designed approximate multipliers, besides offering reduced area/latency and EDP requirements, produce better visual quality outputs and PSNR values than the multiplier proposed in [6]. The approximate multiplier *W*, apparently, produces better PSNR value than those produced by Ca and Cc. However, the input values analysis, in Fig. 12, of the image under consideration shows that most of the multiplications during the image smoothing process are limited to a narrow band and increasing the multiplication output accuracy for this band can increase the accelerator's output quality. Exploiting the asymmetric nature of our proposed multiplier, the mutual swapping of all input values to our approximate multipliers for SUSAN image smoothing accelerator and input-image under consideration results in enhanced output qualities with higher PSNR values as shown in Table 6.


**Figure 11: Accurate and approximate multiplier Ca based accelerator output**

Hence depending upon the input-data and the application under analysis, *Ca, Cc* or $Ca_s$, $Cc_s$ can be deployed for achieving desired area, latency, EDP gains with required output accuracy.

## 6 CONCLUSION

In this paper approximate 4×2 and 4×4 multipliers have been presented as elementary blocks for designing higher order multipliers. To the best of our knowledge, this is the first work that presents FPGA-specific approximate partial product generation and their summation. The generic nature of the presented methodology also opens the door for area-efficient and reduced latency multipliers for future FPGA versions.

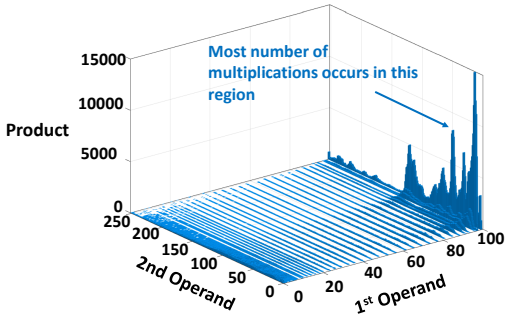



**Figure 12: Analysis of input image: SUSAN application 8×8 multiplication histogram**

**Table 6: PSNR values of 8×8 approximate multipliers**

| Multiplier Architectures | SUSAN Accelerator PSNR |
|---|---|
| Accurate | ∞ |
| Ca | 33.7162 |
| Cc | 25.6022 |
| Approximate 4: W[19] | 47.4939 |
| Approximate 5: K[6] | 17.9443 |
| $Ca_s$ (Ca Swapped Inputs) | 59.1198 |
| $Cc_s$ (Cc Swapped Inputs) | 27.3665 |

## REFERENCES


[1] K. Bhardwaj et al. 2014. Power-and area-efficient Approximate Wallace Tree Multiplier for error-resilient systems. In *ISQED*. IEEE.
[2] N. Brunie et al. 2013. Arithmetic core generation using bit heaps. In *FPL*.
[3] V. K Chippa et el. 2013. Analysis and characterization of inherent application resilience for approximate computing. In *DAC*.
[4] A. K. Verma et al. 2008. Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design. In *DATE*.
[5] M. Shafique et al. 2015. A low latency generic accuracy configurable adder. In *DAC*.
[6] P. Kulkarni et al. 2011. Trading Accuracy for Power with an Underdesigned Multiplier Architecture. In *Internatioal Conference on VLSI Design*.
[7] S. Hashemi et al. [n. d.]. Drum: A dynamic range unbiased multiplier for approximate applications. In *ICCAD*.
[8] V. Gupta et al. 2013. Low-Power Digital Signal Processing Using Approximate Adders. *IEEE Transactions on CAD of Integrated Circuits and Systems* (2013).
[9] V. Gupta et al. 2011. IMPACT: imprecise adders for low-power approximate computing. In *ISLPED*.
[10] Intel. 2017. Integer Arithmetic IP Cores User Guide. (2017). https://www.altera.com/en_US/pdfs/literature/ug/ug_lpm_alt_mfug.pdf
[11] A. B Kahng et al. 2012. Accuracy-configurable adder for approximate arithmetic designs. In *DAC*.
[12] M. Kumm et al. 2015. An efficient softcore multiplier architecture for Xilinx FPGAs. In *ARITH*.
[13] Ian Kuon and Jonathan Rose. 2007. Measuring the gap between FPGAs and ASICs. *IEEE TCAD* 26, 2 (2007).
[14] Chia-Hao Lin et al. 2013. High accuracy approximate multiplier with error correction. In *ICCD*.
[15] C. Liu et al. 2014. A low-power, high-performance approximate multiplier with configurable partial error recovery. In *DATE*.
[16] J. Mody et al. 2015. Study of approximate compressors for multiplication using FPGA. In *IC-GET*.
[17] V. Mrazek et al. 2017. EvoApproxSb: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In *DATE*.
[18] H. Parandeh-Afshar et al. 2011. Measuring and reducing the performance gap between embedded and soft multipliers on FPGAs. In *FPL*.
[19] S. Rehman et al. 2016. Architectural-space exploration of approximate multipliers. In *ICCAD*.
[20] Xilinx. 2011. LogiCORE IP Multiplier v11.2. (2011). https://www.xilinx.com/support/documentation/ip_documentation/mult_gen_ds255.pdf
[21] Xilinx. 2016. 7 Series FPGAs Configurable Logic Block User Guide. (2016). https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf