Area-optimized Accurate and Approximate Softcore Signed Multiplier Architectures

Salim Ullah, Hendrik Schmidl, Siva Satyendra Sahoo, Semeen Rehman and Akash Kumar

Abstract—Multiplication is one of the most extensively used arithmetic operations in a wide range of applications. In order to provide resource-efficient and high-performance multipliers, previous works have proposed different designs of accurate and approximate multipliers—mainly for ASIC-based systems. However, the architectural differences between ASICs and FPGA-based systems limit the effectiveness of these multipliers for FPGA-based systems. Moreover, most of these multiplier designs are valid only for *unsigned* numbers. To bridge this gap, we propose a novel implementation technique for designing resource-efficient and low-power accurate and approximate *signed* multipliers which are optimized for FPGA-based systems. Compared to Vivado's area-optimized multiplier IPs, the designs obtained using our proposed technique occupy 47% to 63% less area (*Lookup Tables*). To accelerate further research in this direction and reproduce the presented results, the RTL and behavioral models of our proposed methodology are available as an open-source library at *https://cfaed.tu-dresden.de/pd-downloads*.

Index Terms—Signed Multiplier, Booth's Multiplication, Accurate, Approximate Computing, FPGA, Energy Efficiency.

1 Introduction

State-of-the-art field-programmable gate arrays (FPGAs), such as Intel Stratix-10 and Xilinx UltraScale+, use hard digital signal processing (DSP) blocks to provide high-performance multipliers and accumulators for a wide range of DSP applications. These DSP blocks are manually optimized, like an application-specific integrated circuit (ASIC), to provide energy and performance gains for different applications. However, the fixed locations and fixed bit-widths of these DSP blocks may result in degraded performance for some applications. We have previously reported the results of two different implementations—with and without DSP blocks—for different DSP applications, such as Nova and Viterbi decoder, in [10]. Our results show that the fixed locations of the allotted DSP blocks results in increasing the routing and critical path delays for some applications. For small applications, it may be possible to improve the critical path delays of the applications by performing manual floorplanning. However, for larger applications with competing resource requirements, such as contentions for DSP blocks and Block RAMs, it might not be possible to take significant advantage of the manual floorplanning for optimizing the overall performance of an application. Further, we have also reported in [10] that the implementation of some applications may consume a large number of the available DSP blocks for performing different arithmetic operations. An exhaustive utilization of DSP blocks, by a single application, will result in utilizing logic-based soft arithmetic blocks for other concurrently running applications (or functions) on the same FPGA. Moreover, the utilization of DSP blocks having M×M multipliers for obtaining Y×Y and Z×Z multipliers, where M>Y and M<Z, can degrade the performance of overall implementations [3]. Similar results about the potential limitations associated with DSP blocks utilization are also reported in [2]. Therefore, it is always advantageous to have low area and high performance logic-based soft multipliers along with DSP blocks, as provided by modern FPGAs vendor, such as Xilinx [4].

Among famous multiplier options, the Wallace [5] and Dadda [6] multipliers have high resource requirements for achieving high performance by parallel summation of partial products. The Booth's multiplication algorithm [7] reduces the number of partial products by encoding multiplier bits to achieve performance gains and area efficiency. The Baugh-Wooley's algorithm [8] focuses on the elimination of sign extension of partial products to obtain low-area and reduced power multipliers for ASICs. Utilizing Booth's algorithm, Kumm et al. have presented an area efficient radix-4 unsigned accurate multiplier implementation for Xilinx FPGAs [9]. Their implementation can also support the multiply-accumulate (MAC) operation. Walters has also used 6-input LUTs to implement signed multipliers for Xilinx FPGAs [30], [31]. However, these implementations do not discuss the possibility of supporting MAC operations. Parandeh-Afshar et al. have used Booth's and Baugh-Wooley's multiplication algorithms for area-efficient multiplier implementation using Altera (now Intel) FPGAs [12]. The authors of [32] have also used the adaptive logic module (ALM) of Intel FPGAs for implementing soft multipliers. However, the implementation results only describe LUTs utilization. The critical path delay and the energy consumption of the implementations have not been discussed.

For a wide range of applications, the exactness of intermediate operations can be compromised without significantly degrading the quality of final output to obtain area, energy and performance gains [13]. Such applications can have inherent resilience to approximations in input data and intermediate operations. Examples of such applications are mostly in the domain of digital signal processing, machine learning and data mining. Error-resilient applications such as deep neural networks have millions of multiply-accumulate operations. For example, Deep Residual Learning (ResNet-152) [33] has 11.3 billions MAC operations per forward pass for the processing of a single image. Therefore, for these applications, approximate multipliers can be utilized for obtaining area-optimized and energy-efficient implementations. Utilizing the inherent error-resilience of such applications, previous works have proposed different approximate multiplier designs. The authors of [14], [15], [20], [21] have proposed different approximate partial products reduction trees for performance and energy gains. Similarly, a method to generate approximate partial products for radix-4 Booth multiplication has been proposed in [28]. Utilizing the concept of modular implementation of multipliers [22], the authors of [16], [17], [18] have presented approximate 4×4 and 2×2 multipliers for generating higher order multipliers. The work in [11] proposed approximate 4×2 and 4×4 multipliers for efficiently utilizing the 6-input lookup tables of modern FPGAs such as Xilinx Ultrascale+. Utilizing different approximate adders and multipliers from literature, an open source library of 8-bit approximate adders and multipliers, EvoApprox8b, has been presented in [19]. Using the dominating design points of EvoApprox8b, a library of FPGA-optimized approximate multipliers—SMApproxLib—has been presented in [10]. However, due to the following limitations, these works cannot be considered for designing approximate *signed* multipliers for FPGA-based systems.

1) The approximation techniques presented in most of these works, such as [17], [18] and [28], ignore the architectural specifications of FPGAs; therefore, these techniques are less effective in gaining ASIC like energy, performance and area gains when used for FPGA-based systems. Fig. 1 shows the comparison of ASIC and FPGA-based implementation results for four multipliers, D1–D4, randomly selected from EvoApprox8b library [19] and an 8×8 approximate multiplier D5 from [17]. These results describe the performance gains of different approximate multipliers with respect to an accurate multiplier implementation. The ASIC-based implementation results for D1–D4 and D5 are obtained from [19] and [17] respectively. For obtaining FPGA-based implementation results, D1–D4 and D5 multipliers are implemented on Kintex-7 FPGA using Xilinx Vivado 17.4 tool. As shown in Fig. 1, the performance gains, reported for ASIC-based implementations, are not proportionally translated for FPGA-based implementations. The architectural differences between ASICs and FPGAs are the main reasons for this

- *S. Ullah, H. Schmidl, S. S. Sahoo and A. Kumar are with Technische Universität Dresden, Germany.*
- *S. Rehman is with Technische Universität Wien, Austria.*

Manuscript received 20 Mar. 2019; revised 29 Mar. 2020; accepted 5 Apr. 2020.
(Corresponding author: Salim Ullah.)
Recommended for acceptance by G. Constantinides.
The work presented in this article is supported by the German Research Foundation (DFG) funded project ReAp (Project Number: 380524764).
Digital Object Identifier no. 10.1109/TC.2020.2988404

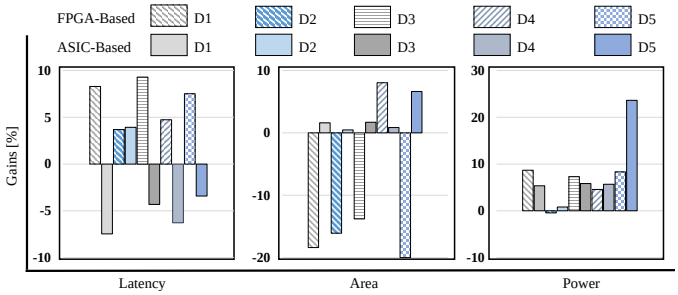


Fig. 1. Implementation results of five state-of-the-art approximate multipliers on ASICs and FPGAs

dissimilar performance gains for FPGA-based implementations. FPGAs consist of lookup tables (LUTs) and carry chains. Any approximation techniques, optimized for FPGA-based systems, must consider the structure and configurations of these LUTs and carry chains.

2) Most of these techniques are for unsigned numbers only. For example, the hundreds of approximate multipliers presented in EvoApprox8b [19] and SMApproxLib [10] focus on unsigned numbers only. The 4×2 and 4×4 approximate multipliers in [11] and the 2×2 approximate multiplier modules used in [17] and [18] are for unsigned numbers only. The inaccurate 4:2 counter [16] and the partial products reduction techniques in [14], [15], [20] and [21] do not discuss the applicability of their approaches for signed numbers. Therefore, these techniques cannot be directly used for the approximate multiplication of signed numbers.

3) The modular approach of designing bigger multipliers using smaller multipliers, used by [11], [16], [17], [18], is advantageous for designing smaller multipliers only. Our empirical results show that this technique consumes more FPGA resources for higher bit-width multipliers.

Our Novel Contributions

To address the above limitations of state-of-the-art multipliers, we present a novel methodology for implementing accurate and approximate signed array-multipliers for FPGA-based systems. Fig. 2 presents an overview of the different stages of our proposed methodology. Our contributions include:

- *Accurate signed multiplier implementation*: Utilizing the 6-input LUTs and associated fast carry chains of modern FPGAs, we present an area-optimized implementation of radix-4 accurate Booth multiplier—that we refer to as *Booth-Mult*. The proposed M × N *Booth-Mult* further supports the addition of an $M - bit$ number to provide a sort of *Multiply-Accumulate* (MAC) functionality. We then analyze *Booth-Mult* and perform FPGA-specific area optimizations to propose *Booth-Opt* multiplier with further reduction in the overall area of the multiplier. Our implementation of a 24×24 *Booth-Opt* multiplier offers 47% reduction in the area when compared with the area-optimized multiplier IP provided by Vivado [4].
- *Analysis for possible approximation*: We present an analysis of our *Booth-Opt* multiplier for identification of possible venues for approximations to achieve area, energy and performance gains. One of the challenges during this exploration is the preservation of correct sign-bit in the final product to reduce the maximum output error. Based on the work of [23], we identify the starting elements of each partial product row as possible positions for approximations. This choice ensures the preservation of correct sign-bit in the final product during multiplication by reducing the propagation of erroneous carry bits to higher order bits.
- *Approximate signed multiplier*: Based on our analysis, we then present an approximate signed array multiplier *Booth-Approx* with further area, energy and performance gains. When compared with the Vivado area-optimized multiplier IP, the proposed *Booth-Approx* implementation of a 24×24 signed multiplier provides 49% and 38% gains in terms of resource utilization and energy consumption, respectively. To the best of our knowledge, this is the first attempt towards implementation of FPGA-based approximate signed array multipliers.
- *High-level application testing environment*: We provide a high-level application environment for testing the efficacy of different accurate and approximate arithmetic components. Our application environment utilizes *Genetic Algorithm-based multi-objective* design space exploration, and it is applicable for convolution based signal processing applications. Utilizing the environment, we have tested our accurate and approximate multipliers for Gaussian Image Smoothing application and evaluated the trade-off between output quality and area reductions. The approximate multiplier-based implementation of Gaussian Image Smoothing application results in up to 57.9% reduction in resource utilization, with minimal degradation in the image output quality, when compared with the Vivado's area-optimized multiplier IP-based implementations.

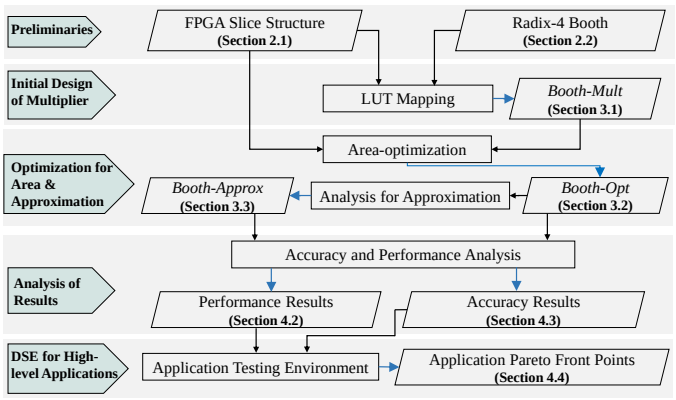


Fig. 2. Overview of proposed methodology

The rest of the paper is organized as follows: Section 2 briefly discusses the preliminaries required for understanding the paper, followed by the proposed methodology in Section 3. Section 4 discusses the experimental setup, implementation results and behavioral outputs for real-world applications. Finally, Section 5 concludes the paper.

2 Preliminaries

2.1 Xilinx FPGA Slice Structure

The configurable logic block (CLB) is the main computational block of FPGAs for implementing any kind of circuits on FPGAs. The CLB of a modern Xilinx FPGA, such as Xilinx UltraScale+, consists of one slice having eight 6-input lookup tables (referred to as LUT6_2), 8-bit long carry chain and sixteen flip-flops [24]. The same resources are arranged in two slices in a Xilinx 7 series FPGA [25]—which has been used for all implementations in this paper. As shown in Fig. 3, a LUT6_2 can be used to implement either a single 6-input combinational circuit or two 5-input combinational circuits. For the configuration of LUT6_2, a 64-bit INIT value is assigned to it. This INIT value denotes all the input combinations of LUT6_2 for which a "1" is received at the output. Utilizing O5 as carry-generate signal and O6 as the carry-propagate signal, a carry-lookahead adder can be implemented using the associated carry chain. However, O5 can be bypassed by the external IX signal for providing the carry-generate signal. The input carry, "CIN", can be either assigned to constant '0/1' or to "COUT" of another carry chain from a different slice.

2.2 Booth's Multiplication Algorithm

Booth's multiplication algorithm reduces the number of partial products to enhance the performance of a multiplier. A radix-4 Booth's multiplier halves the total number of partial products for an M × N signed multiplier. Equation (1) shows the 2's

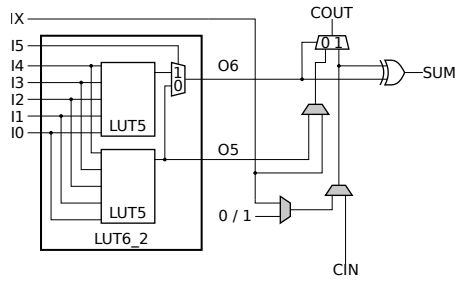


Fig. 3. Xilinx 6-input lookup table and associate carry chain

TABLE 1
Booth's Encoding

S.No.	b_{n+1}	b_n	b_{n-1}	BE	s	c	z
0	0	0	0	0	0	0	1
1	0	0	1	1	0	0	0
2	0	1	0	1	0	0	0
3	0	1	1	2	1	0	0
4	1	0	0	$\bar{2}$	1	1	0
5	1	0	1	$\bar{1}$	0	1	0
6	1	1	0	$\bar{1}$	0	1	0
7	1	1	1	0	0	0	1

complement representations of *multiplicand A* and *multiplier B*, and the corresponding radix-4 booth's multiplication is summarized in (2).

$$\begin{aligned} A &= -a_{M-1}2^{M-1} + \cdots + a_2 2^2 + a_1 2^1 + a_0 \\ B &= -b_{N-1}2^{N-1} + \cdots + b_2 2^2 + b_1 2^1 + b_0 \end{aligned} \tag{1}$$

$$\begin{aligned} A \cdot B &= \sum_{n=0}^{N/2} B \cdot BE_{2n} 2^{2n} \\ where \quad BE_{2n} &= -2a_{2n+1} + a_{2n} + a_{2n-1} \end{aligned} \tag{2}$$

The values of Booth's encoding (BE) in (2) are in the range of $\pm 0, \pm 1, \pm 2$ and can be computed as shown in Table 1. These values are computed by LUT6_2s in our proposed designs. A partial product is shifted left if BE = 2 (denoted by s = 1). Similarly, for a negative value of BE (denoted by c = 1), the 2's complement of the corresponding partial product is calculated by initially taking 1's complement of the partial product and adding a '1' to the LSB position. For BE = 0 (denoted by z = 1), the corresponding partial product is replaced by a string of zeros.

Our proposed implementation and the implementations presented by Kumm [9] and Walters [30], [31] have utilized similar configurations of the 6-input LUTs to realize the radix-4 booth-encoding. However, Kumm has considered unsigned numbers only and, therefore, has used a different configuration

TABLE 2
Sign Extension for Booth's Multiplier

b_{n+1}	b_n	b_{n-1}	BE	MSB Multiplicand	SE
0	0	0	0	0	0
0	0	0	0	1	0
0	0	1	1	0	0
0	0	1	1	1	1
0	1	0	1	0	0
0	1	0	1	1	1
0	1	1	2	0	0
0	1	1	2	1	1
1	0	0	$\bar{2}$	0	1
1	0	0	$\bar{2}$	1	0
1	0	1	$\bar{1}$	0	1
1	0	1	$\bar{1}$	1	0
1	1	0	$\bar{1}$	0	1
1	1	0	$\bar{1}$	1	0
1	1	1	0	0	0
1	1	1	0	1	0

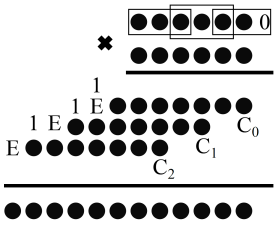


Fig. 4. Sign extension of partial products in Booth's multiplier [26]

for the most significant LUTs in each partial product row. Ours and Walters' designs have considered signed numbers, and the most significant LUTs are responsible for computing the correct sign of a partial product row. However, both implementations have used different LUT configurations. The technique used by our implementation is discussed in Section 2.3. Moreover, these state-of-the-art works do not present any analysis for possible approximations to obtain performance gains. We present a methodology for obtaining approximate multipliers, the required LUTs configurations for approximations, and an in-depth analysis of these approximations on the output accuracy.

2.3 Sign Extension for Booth's Multiplier

All the partial products, in a signed multiplication, must be properly sign extended before reducing them to a final product. As shown by rows number 4, 5 and 6 in Table 1, the Booth's encoding can result in negative partial products. However, the correct sign of a partial product in Booth's multiplier depends on the BE and the MSB of the multiplicand. If the MSB of the multiplicand is '0' and the BE is either a positive number or '0', the partial product will be 0-extended. Similarly, if the MSB of the multiplicand is '1' and the BE is a non-zero positive number, the partial product will be extended with '1'. A complete list of all possible sign extension cases, denoted by 'SE', is presented in Table 2. Utilizing the method presented by Bewick in [26], Fig. 4 shows an efficient technique for sign extension in 6×6 Booth's multiplier. The C_0, C_1 and C_2 will be '1' for negative partial products to represent them in 2's complement notation. The 'E' bits are the complement of corresponding 'SE' bits in Table 2. This technique significantly reduces the sign extension of each partial product row to a maximum of two more bit-positions.

Our proposed accurate (*Booth-Mult* and *Booth-Opt*) and approximate (Booth-Approx) implementations do not solely depend on FPGA synthesis and implementation tools for the optimization of multipliers. We efficiently utilize the characteristics of LUT6_2s and associated carry chains for the proposed implementations. The accurate multiplier design presented by Kumm in [9] underutilizes the least significant LUTs in each partial product row of a multiplier. Our proposed LUT configurations efficiently utilize the LUT6_2s to reduce the total number of utilized LUTs. Moreover, the designs presented in [9], [30], and [31] have used a separate LUT6_2, at the most significant bit location of each partial product row, for forwarding the sign extension information to succeeding partial product row. The design presented in [31] has shown this LUT location by providing a constant '1' to the carry chain. However, our proposed accurate multiplier *Booth-Opt* does not require such configurations and reduces the total LUTs utilization. The proposed *Booth-Approx* further improves the utilization of the configurable logic block.

3 Proposed Methodology

Utilizing the concepts of Booth's multiplication (Section 2.2) and the efficient sign extension technique (Section 2.3), we present our technique for optimization of resource utilization and energy efficiency for both accurate and approximate signed multipliers. The Booth's encoding scheme, shown in Table 1, is implemented by the LUT configuration *type-A* shown in Fig. 5(a). It receives six inputs i.e. a_m, a_{m-1} (from multiplicand), b_{n+1}, b_n, b_{n-1} (from multiplier) and pp_{in} (partial product sum from previous row). Depending upon the shift flag 's', either a_m or a_{m-1} will be forwarded. Similarly, depending upon the complement flag 'c', the 1's complement of a partial product

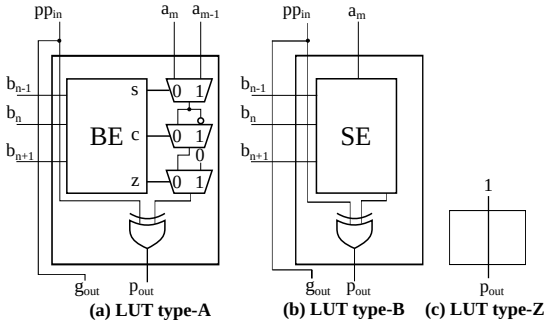


Fig. 5. Configuration of LUTs used in proposed methodology

can be forwarded. The third MUX, controlled by zero flag 'z', can make partial product zero if the 'z' flag =1. The output of the third MUX is XORed with the partial sum (pp_{in}) and forwarded to associated carry chain as carry propagate signal. The carry generate signal for the carry chain is provided by the pp_{in}.

The sign extension of each partial product is implemented by the LUT configurations *type-B* and *type-Z*, shown in Fig. 5(b) and (c) respectively. LUT type-B performs XOR operation between the SE and pp_{in} signals. The output of this operation is forwarded as propagate signal to the carry chain. The pp_{in} is also used as the carry-generate signal. LUT type-Z represents the most significant constant '1' in a partial product row, as shown in Fig. 4 and is used for forwarding the sign extension information to higher order partial product rows using sum and carry output bits of the carry chain.

3.1 Accurate Signed Multiplier: Booth-Mult

Utilizing LUTs of type A, B and Z, Fig. 6 shows the implementation of a 6×6 signed multiplier. As described in Fig. 4, a C_x is added at the LSB position of each partial product for representing it in 2's complement format. This task of finding the correct C_x is performed by the rightmost LUT in each partial product row in Fig. 6. This carry will be used by the carry chain element of next LUT of type-A. The most significant two LUTs, LUT type-B and LUT type-Z, in each partial product row are responsible for implementing correct sign extension. LUT type-B computes the correct sign bit, and the LUT type-Z is used for adding the constant '1' as shown in Fig. 4. However, the last partial product row does not contain a LUT of type-Z. Due to the very regular pattern of our proposed multiplier implementation, the LUTs required for implementing an M×N multiplier can be estimated by (3), where 'M' is the multiplicand and 'N' is the multiplier. Since 'N' defines the number of partial product rows in an implementation, mutual swapping of multiplicand and multiplier for, $Multiplicand < Multiplier$, can result in a more resource efficient design. As shown in Fig. 6, the 'pp_{in}' signals of LUT type-A have been initialized to constant '0' in the first partial product row. For an M×N implementation of the proposed multiplier, an M-bit number can be further added using these 'pp_{in}' signals of the first partial product row to achieve the MAC operation. Since digital signal processing applications frequently utilize MAC operations, our proposed accurate multiplier can be very useful for such applications to obtain significant area gains.

$$LUTs\ for\ M \times N\ multiplier\ = (M+4) \times \left\lceil \frac{N}{2} \right\rceil - 1 \qquad (3)$$

3.2 Area Optimized Accurate Signed Multiplier: Booth-Opt

The analysis of the implementation shown in Fig. 6 reveals the following observations:

- The first two LUTs in each partial product row are underutilized. The first LUT has three constant '0' inputs, and only the carry output of its associated carry chain is used. Similarly, the second LUT has also a constant '0' input. It is possible to achieve the functionalities of these two LUTs, in each partial product row, using a single modified LUT shown in Fig. 7(a). The non-shaded MUXes and XOR gate perform similarly to those in LUT type-A. However, the shaded MUXes are responsible for generating and forwarding the correct carry to the next LUT type-A in the same partial product row. The generated carry is also used for producing the least significant product bit, of the respective partial product row, using the shaded XOR gate.
- LUT type-Z, in each partial product row, is used for forwarding sign extension information to other partial product rows. It forwards a constant '1' as the carry-propagate signal to the associated carry chain. As shown in Fig. 3, this will result in SUM = $\overline{CIN}$ and COUT = CIN. The $\overline{CIN}$ is then used by a LUT type-A in the succeeding partial product row. However, instead of using LUT type-Z for generating $\overline{CIN}$, the LUT type-A can be modified to invert an incoming signal internally. This results in LUT type-A2 shown in Fig. 7(b).

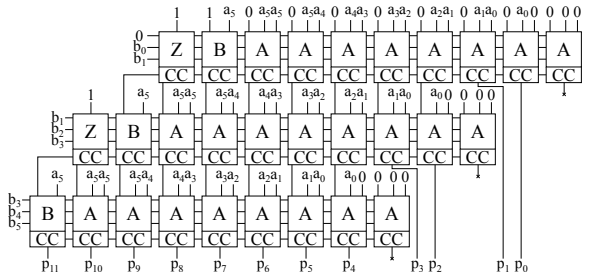


Fig. 6. A 6×6 accurate multiplier implementation (Booth-Mult)

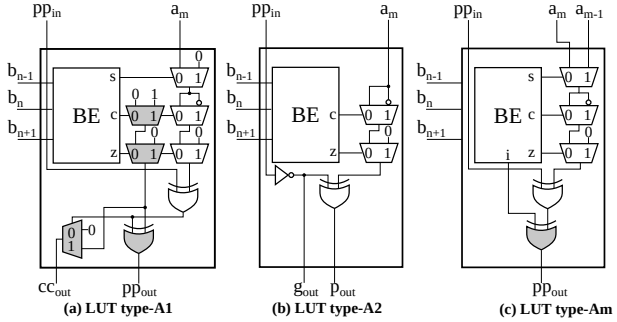


Fig. 7. Modified configuration of LUTs

Utilizing LUTs type-A1 and type-A2, Fig. 8, shows an area-optimized implementation of a 6×6 accurate signed multiplier. Each partial product row starts with LUT type-A1. The LUT type-A2, in first partial product row, with inputs a_5 and constant '1' is identical to LUT-type A with inputs a_5 and constant '0'. The LUT type-A in first partial product row is replaced with LUT type-A2 for making the first partial product row identical to other partial product rows. A carry out of carry chain element associated with a LUT type-B is forwarded to LUT type-A2 and type-B in succeeding partial product row. For this area optimized multiplier implementation, the total number of LUTs required for implementing an M×N multiplier is represented in (4).

$$LUTs\ for\ M \times N\ multiplier\ = (M+2) \times \left\lceil \frac{N}{2} \right\rceil \qquad (4)$$

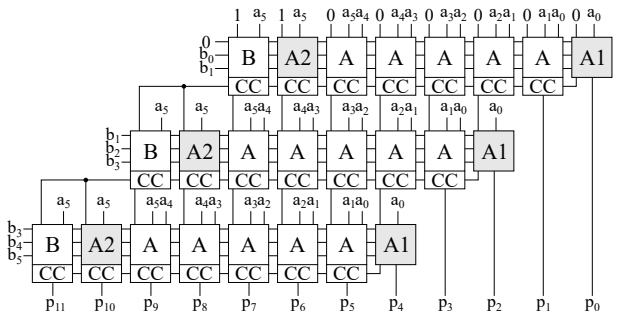


Fig. 8. A 6×6 area-optimized accurate multiplier (Booth-Opt)

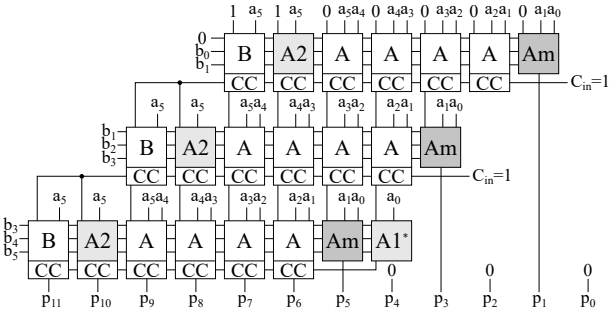


Fig. 9. A 6×6 approximate multiplier implementation (Booth-Approx)

3.3 Approximate Signed Multiplier: Booth-Approx

Utilizing the proposed area optimized accurate signed multiplier—Booth-Opt— as a base architecture, we perform a detailed analysis of the possible trade-offs between final output accuracy, latency and energy gains. For describing the accuracy of the final product of the approximate multipliers, we have used a variety of quality metrics, such as the number of error occurrences, maximum error magnitude, average error, average relative error, and the normalized mean error distance. These quality metrics are commonly used in the literature for the characterization of the approximate arithmetic circuits [11], [17], [18], [19]. Multiple uniform distributions of all input combinations for an N×N multiplier, are used to estimate the power dissipation in each of the instantiated LUTs of the implementation. Our analysis reveals that the first two LUTs in each partial product row contribute more to the dynamic power consumption and critical path delay of the multiplier. For instance, for an 8×8 multiplier the highest power dissipation of $170\mu W$ is observed for the first LUT compared to $84\mu W$ for the fourth LUT in the fourth partial product row. Further, three of the four first-placed LUTs in the partial product rows contribute to all of the top five worst critical path delays. Therefore, approximating the functionalities of the first two LUTs in each partial product row can lead to significant power and latency gains. We recommend the following modifications/suggestions to an N×N base architecture for achieving a latency and power-optimized approximate signed multiplier.

- We propose truncation of the first LUT, for the LSB, in each partial product row of an N×N multiplier to static '0'. This truncation results in a significant decrease in dynamic power consumption.
- To approximate the output of second LUT in each partial product row, we propose LUT configuration type-Am, shown in Fig. 7(c). LUT type-Am does not use the associated carry chain and predicts the missing input carry, using signal 'i', to generate an approximate output. The detailed error-analysis of the outputs generated by second LUTs in each partial product row of our base architecture, reveals that in the absence of an input carry, most errors are generated for the Booth's Encoding $\overline{1}, \overline{2}, 2$. To reduce the number of these wrong outputs, LUT type-Am predicts the missing input carry as constant '1' for Booth's Encoding $\overline{1}, \overline{2}, 2$ and uses it for computing the approximate output.
- As shown by [23], the chances of errors in higher order output bits, produced by an initial incorrect input carry, decreases with the increasing length of the carry chain. Our proposed approach recommends a constant '0/1' input carry to the first LUT of type-A in a partial product row. The exhaustive error analysis of a 8×8 multiplier reveals that providing a constant '1' as input carry results in a decreased relative error in the final output. However, as the most significant partial product row has the maximum contribution in the accuracy of the final product, the carry generation in the most significant partial product row should remain unaffected. Therefore, the most significant partial product row utilizes a LUT type-A1 (denoted by A1*) only for carry generation and the pp_{out} signal of A1* is truncated to constant '0'.

Utilizing these guidelines, the architecture of a 6×6 approximate signed multiplier is presented in Fig. 9. The resource utilization for our proposed approximate signed multiplier can be estimated by (5).

$$LUTs\ for\ M \times N\ multiplier\ = (M+1) \times \left\lceil \frac{N}{2} \right\rceil + 1 \quad (5)$$

4 Results and Discussion

4.1 Experimental Setup

We have used VHDL, Xilinx Vivado 17.4 and XC7V585T device of Virtex-7 FPGA (unless stated otherwise) for the implementation of all presented multipliers. Vivado Simulator and Power Analyzer tools have been used for the calculation of dynamic power values. We have compared our proposed accurate signed multiplier with the accurate multiplier IPs provided by Xilinx Vivado [4], "$S4$" [9], "$S6$" [30], "$S7$" [31], and "$S8$" [32]. Further, the proposed signed approximate multiplier is compared with the implementations presented in "$S1$" [17], "$S2$" [18], "$S3$" [11], a randomly selected 8×8 multiplier from "$S5$" [19]* and precision-reduced "Trunc" multipliers. For an M × N "Trunc" multiplier, the two LSBs of each input operand have been truncated, and an (M-2) × (N-2) multiplier has been used to implement a precision-reduced multiplier. This technique is different from two other possible design approaches – (1) truncating the four LSBs in the final product of an accurate M × N multiplier to '0' after performing multiplication. This method does not provide any reduction in overall resource utilization, critical path delay, and energy consumption. (2) Removing the logic for the computation of the four LSBs in the final product of an M × N multiplier, and generating the rest of the product bits ($M + N - 4$ MSBs) accurately. For example, to implement accurate 12 MSBs in this second design for an 8×8 multiplier, 353 LUTs are utilized—an increase of 301% over the 88 LUTs of Vivado's area optimized 8×8 design.

For the *unsigned* multipliers presented in [9], [11], [17], [18] and [19], we have shown implementation results with/without implementing signed-unsigned converters. These converters have been used to provide *2's complement signed numbers* to the *unsigned multipliers*. To produce precise area (LUTs) utilization, critical path delay (CPD) and dynamic power consumption values, we have implemented each design multiple times with different timing constraint. In each iteration of the design implementation, the critical path is adjusted according to the worst negative slack from the previous iteration. For this article, we have fixed the maximum number of iterations to 10. The accuracy of the proposed approximate multiplier has been computed for multiple uniform distributions of all input combinations. Moreover, the C and Python-based behavioral models of proposed accurate/approximate multipliers are also deployed in Gaussian Image Smoothing application and an artificial neural network (ANN) for testing the effects of the proposed multipliers in the real-world application.

4.2 Performance Characterization of Proposed Multipliers

Table 3 shows the comparison of the resource utilization, critical path delay and energy consumption of the proposed accurate (Booth-Opt) and approximate (Booth-Approx) multipliers with different state-of-the-art accurate and approximate multipliers. To compare Booth-Opt with *"S6"* and *"S7"* and *"S8"*, we have used the implementation results presented in the respective articles [30], [31] and [32] respectively. For *"S6"* and *"S7"*, the total number of LUTs used has been computed considering the carry chains required for implementing the designs. Booth-Opt is more resource-efficient than *"S6"* across different bit-widths. For example, compared to the 12×12 *"S6"*, Booth-Opt offers a 13% reduction in the total utilized LUTs. *"S7"* has the same LUTs utilization as offered by our proposed Booth-Opt implementation. However, the implementations in [30] and [31] have used a target design period of $1ns$, and the corresponding results show that none of the implementations meets the target design period. Nonetheless, to compare the performance of our Booth-Opt design with *"S6"* and *"S7"*, we have used the

*A generic and open-source implementation for every size of multiplier was not available.

TABLE 3
Implementation results of different multipliers. The 'S1', 'S2', 'S3', 'S4' and 'S5' multipliers are implemented with the signed-unsigned converters. The CPD and PDP are in ns and pJ, respectively. The shaded rows show approximate designs.

Design	4×4			6×6			8×8			12×12			16×16			24×24		
	LUTs	**CPD**	**PDP**	**LUTs**	**CPD**	**PDP**	**LUTs**	**CPD**	**PDP**	**LUTs**	**CPD**	**PDP**	**LUTs**	**CPD**	**PDP**	**LUTs**	**CPD**	**PDP**
Booth-Opt.	12	2.15	1.09	24	3.09	2.67	40	4.25	5.14	84	6.31	11.96	144	7.64	21.15	312	11.37	49.17
Booth-Approx.	11	1.94	0.81	22	2.64	2.18	37	3.41	4.22	79	5.30	10.17	137	6.88	19.14	301	10.99	48.26
S1: Rehman [17]	18	2.23	0.86	49	4.82	3.78	92	4.99	7.10	228	6.98	20.80	404	7.03	22.32	895	9.43	101.63
S2: Kulkarni [18]	20	2.12	0.87	52	4.83	4.91	86	4.89	7.42	189	6.37	20.77	330	6.59	20.39	777	9.45	97.48
S3: Ullah [11]	22	3.34	1.19	46	5.03	3.98	81	5.19	7.41	185	7.11	20.39	296	7.33	18.58	697	9.69	92.35
S4: Kumm [9]	24	3.84	2.00	49	5.10	6.87	73	6.08	9.69	138	7.65	21.65	217	9.52	32.93	427	13.38	85.66
S5: Mrazek [19]	-	-	-	-	-	-	110	4.43	9.75	-	-	-	-	-	-	-	-	
Trunc	2	0.659	0.056	23	1.576	1.21	43	2.15	3.06	102	3.52	8.97	214	4.107	14.76	514	6.07	53.97
S6: Walters [30]	-	-	-	-	-	-	43	-	-	97	-	-	155	-	-	-	-	-
S7: Walters [31]	-	-	-	24	-	-	40	-	-	84	-	-	144	-	-	312	-	-
Vivado speed	18	2.14	1.06	41	3.43	3.26	74	3.54	5.73	162	4.20	19.79	286	4.27	34.35	627	5.98	77.25
Vivado area	30	2.91	2.25	47	3.39	4.73	88	3.45	9.07	175	5.00	15.33	326	5.04	35.25	592	5.55	78.41

critical path delays of these designs normalized to the critical path delay of Vivado speed-optimized IPs. For *"S6"* and *"S7"*, the normalized values have been acquired from [30] and [31]. Booth-Opt provides better overall performance than *"S7"* by offering a 5.4% reduction in the average normalized critical path delay across different sizes of multipliers. Similarly, for comparison with the Intel FPGA-based unsigned design *"S8"*, we compare the reductions in resource utilization of *"S8"* over the Intel Megafunction IP with Booth-Opt over Vivado speed-optimized IP. Booth-Opt provides better resource utilization than *"S8"* for all sizes of multipliers. Moreover, *"S8"* provides reductions in resource utilization for only some sizes of multipliers. For example, for 4×4 and 8×8 multipliers, *"S8"* offers 27% and 0% reductions in resources, respectively, whereas Booth-Opt offers 33% and 46% reductions for the two sizes of multipliers, respectively.

The results in Table 3 also incorporate the signed-unsigned converters for the unsigned multipliers "$S1$", "$S2$", "$S3$", "$S4$" and "$S5$". As discussed previously, the "$S5$" multiplier has been implemented for obtaining only 8×8 approximate multiplication, and hence only a single point is shown for it in Table 3. As shown by the results, the proposed Booth-Opt and Booth-Approx always require less number of LUTs than other state-of-the-art accurate and approximate multipliers for different sizes of multipliers. The area reductions with respect to the Vivado's area-optimized multiplier IPs vary between 47% (for 24×24) and 63% (for 4×4). For example, compared to the 8×8 multiplier IP, the proposed Booth-opt and Booth-Approx show 54.6% and 57.9% area reduction respectively. Similarly, compared to the 24×24 accurate "$S4$" multiplier, the proposed Booth-Opt offers 27% area reductions.

Compared to the state-of-the-art approximate multipliers and Vivado's area/speed optimized-IPs, the proposed accurate and approximate multipliers offer comparable critical path delays, as shown in Table 3. The small increase in the critical path delays of the proposed multipliers is due to the sequential computation of booth-encoded partial products as discussed in Section 3.2. However, the proposed accurate and approximate multipliers always offer reduced critical path delays than accurate "$S4$" multiplier. Compared to the 8×8 "$S4$" multiplier, Booth-Opt and Booth-Approx offer 30% and 44% reduction in critical path delays respectively.

The energy consumption of the proposed multipliers has been compared to other multipliers using Power-Delay product (PDP). It can be observed from Table 3 that the proposed multipliers offer better energy efficiency across different sizes of multipliers. The proposed 24×24 Booth-Opt and Booth-Approx show 37% and 38% reduction in energy when compared to Vivado's area-optimized IP. Similarly, compared to the 24×24 "$S4$" multiplier, our Booth-Opt offers 42.5% reduction in energy.

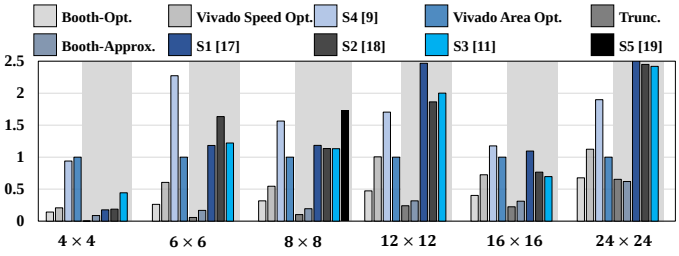


Fig. 10. Products of normalized performance metrics. Values are normalized to Vivado area-optimized multiplier IP. A smaller value reflects better performance. Approximate designs have shaded backgrounds.

TABLE 4
Comparison of implementation results of proposed multipliers with unsigned multipliers

Design	8x8			16x16		
	Area [LUTs]	CPD [ns]	PDP [pJ]	Area [LUTs]	CPD [ns]	PDP [pJ]
Booth-Opt	40	4.25	5.14	144	7.64	21.15
Booth-Approx	37	3.41	4.22	137	6.88	19.14
S1 [17]	57	3.13	4.70	377	4.57	24.03
S2 [18]	80	2.33	5.13	294	4.06	21.73
S3 [11]	57	3.13	4.70	245	5.02	20.77
S4 [9]	51	3.87	6.26	167	6.87	23.57

To highlight the efficacy of our proposed accurate and approximate multipliers, Fig. 10 shows the product of normalized values of total utilized LUTs, CPD, and PDP for each design across different bit-widths. All values have been normalized to the corresponding values of Vivado area-optimized multiplier IP. A smaller value of the product (LUTs × CPD × PDP) presents an implementation with a better performance. The proposed Booth-Opt provides better performance than state-of-the-art accurate multipliers. Although for smaller designs, the *"Trunc"* multiplier performs better than Booth-Approx, the performance gains do not scale proportionally for higher-order *"Trunc"* multipliers. For example, in 24×24 multipliers, Booth-Approx provides a 5.2% reduction in the product of the normalized performance metrics compared to the *"Trunc"* multiplier. Moreover, the error analysis of the approximate multipliers, presented in the next subsection, shows the lower accuracy of *"Trunc"* multipliers across all error metrics.

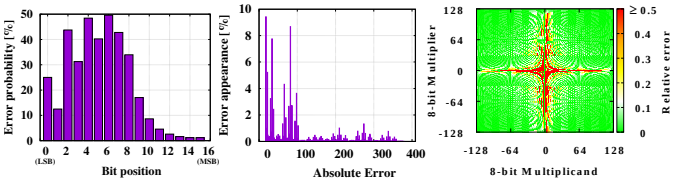
We have also compared proposed designs with the state-of-the-art *unsigned* designs "$S1$", "$S2$" "$S3$" and "$S4$" without using *signed-unsigned converters* for them. As shown by the results in Table 4, our proposed accurate and approximate *signed* designs still offer better area reductions than all other implementations. The unsigned implementations have slightly reduced critical path delays than the proposed multipliers. However, the energy consumption of Booth-Opt and Booth-Approx is still better than many of the state-of-the-art designs. For example, compared to the *unsigned* 16×16 "$S4$" multiplier, the Booth-Opt and Booth-Approx offer 10% and 19% energy reductions respectively. Similarly, Booth-Opt and Booth-Approx offer 12% and 20% reductions in energy consumption, respectively, when compared with the 16×16 unsigned multiplier "$S1$".

4.3 Error Analysis of Proposed Approximate Multiplier

Table 5 presents the error analysis of the proposed approximate multiplier along with precision-reduced "Trunc" and other state-of-the-art approximate multipliers (using signed-unsigned converters). Since the number "-128" cannot be represented using sign-magnitude format for 8×8 multipliers with 8-bit operands, the observed *maximum error* magnitude

TABLE 5
Error analysis of 8×8 approximate multipliers

Design	Error Occurrences %	Maximum Error	Average Error	Max. Relative Error	Avg. Relative Error	NMED
Booth-Approx	90.56	361	85.01	6	0.091	0.0051
$S1$ [17]	86.46	7225	1842.44	1	0.362	0.112
$S2$ [18]	34.19	882	118.875	1	0.0223	0.0072
$S3$ [11]	8.42	2312	101.94	1	0.0121	0.0062
$S5$ [19]	84.43	544	127.11	5.6	0.049	0.0077
Trunc (6×6)	93	759	149.78	15	0.121	0.0091

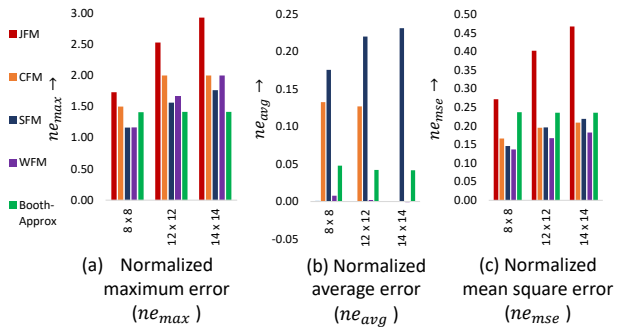


(a) (b) (c)

Fig. 11. Error probabilities in individual product bits (a) Inaccuracy bit histogram (b) PMFs of error values (c) Relative error distribution

in "$S1$", "$S2$", "$S3$" and "$S5$" is 16384. However, to show a fair comparison, the 8-bit operands' range for computing the *maximum error* is limited to $[-127, +127]$ for designs in "$S1$", "$S2$", "$S3$" and "$S5$". As shown by the highlighted cells in the table, the Booth-Approx has the least maximum error magnitude, average error and normalized mean error distance (NMED) among all presented multipliers. Further, it can be observed that Booth-Approx is better than the "Trunc" multiplier across all the presented error parameters. To further explore the error occurrences of the proposed approximate multiplier, Fig. 11 presents the probabilistic error analysis for an 8×8 Booth-Approx multiplier. These results have been obtained for a uniform distribution of all input combinations. As shown by the bit inaccuracy histograms in Fig. 11(a), the probability of errors in individual product bits reduces for higher order product bits. The probability mass functions (PMF) of errors, depicted in Fig. 11(b), also show that the majority of occurred errors have small values. This is also verified by the relative error distribution plot shown in Fig. 11(c). As shown by the results, most of the final products have very small relative errors (on average less than 0.1). This behavior is in accordance with our design modifications discussed in Section 3.3. Since higher order multipliers have long carry chains, therefore, the errors generated by incorrect input carries diminish for higher order product bits. Moreover, a constant '0' multiplicand/multiplier results in an accurate '0' result.

Further, Fig. 12 shows the error metrics of our proposed design compared to that of truncated and truthfully rounded multipliers. The maximum, average and mean squared error of each design is normalized with 2^n, 2^n and 2^{2n} respectively– n being the input bit-width [34]. As seen in the figure, our proposed design outperforms the other designs for normalized maximum error for larger designs. The faithful rounding, after truncation, usually involves some form of compensation to reduce the error [38]. This additional compensation logic is



(a) Normalized maximum error (ne_{max}) (b) Normalized average error (ne_{avg}) (c) Normalized mean square error (ne_{mse})

Fig. 12. Comparison with truncated and rounded multipliers: JFM [35], CFM [36], SFM [37], WFM [34]

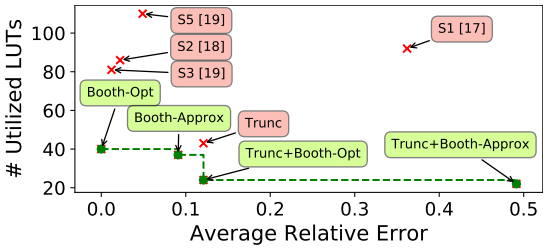


Fig. 13. Design space showing the area-accuracy trade-off of different designs for an 8×8 multiplier (Pareto points are in green).

TABLE 6
Comparison of $all-accurate$ and $all-approximate$ multipliers-based FIR filters

Convolution window size	Relative area reduction (w.r.t. Vivado's IP in %)		PSNR		SSIM	
	Booth-Opt	Booth-Approx	2D	2S	2D	2S
3 x 3	54.6	57.95	50.50	45.72	0.98	0.96
5 x 5	54.6	57.95	51.85	45.19	0.99	0.97
7 x 7	54.6	57.95	52.36	47.55	0.99	0.95

optimized for ASIC-based implementation and can result in large overheads in FPGAs. For instance, the implementation of the compensation logic used in [38] results in 90 LUTs being used for a 8×8 multiplier.

Fig. 13 shows the area and accuracy trade-offs for some of the multiplier implementations discussed in this article. In addition to truncation of two LSBs (*Trunc*) and the related state-of-the-art approximate implementations, we show two design points which are a combination of *Trunc* and the proposed Booth-based designs–*Trunc+Booth-Opt* and *Trunc+Booth-Approx*. As seen in the figure, the Pareto front in this design space comprises of the implementations based on our proposed designs.

4.4 High-level Application Testing

The proposed signed multiplier architectures were used in the implementation of Finite Impulse Response (FIR) filter for image processing applications. Typical hardware realization of FIR filters involves the implementation of N multipliers and N adders—N being the number of *taps* in the filter—to implement convolution. For our current work, we have used *Gaussian Smoothing* as a test case for evaluating the efficiency of using the proposed signed multipliers. Gaussian smoothing of an image involves 2-dimensional (2D) convolution of the image with a *gaussian-kernel*. This 2D convolution can also be achieved by a 2-stage (2S) method that entails successive one-dimensional (1D) convolutions along each of the two directions—horizontal and vertical. These two methods can have large differences in the resource utilization of their realizations—$O(n+m)$ and $O(nm)$ for a window size of $n \times m$ for 2S and 2D respectively.

We performed experiments for convolution window sizes of 3×3, 5×5 and 7×7, and compared the effects of using accurate and approximate signed multipliers (8×8) on the resource utilization and the degradation in processed images quality. We have used the resource utilization—in terms of $LUTs$ used for the multipliers only—of Vivado's area-optimized multiplier IP-based implementation and the corresponding output image quality as the baseline for comparison. Two metrics were used for processed image quality – (1) *PSNR*, an estimation of the noise component in the image, and (2) *SSIM-index*, a measure of the structural similarity between the two images.

TABLE 6 shows the comparison results for Gaussian smoothing using *all-accurate* (Booth-Opt multipliers only) and *all-approximate* (Booth-Approx multipliers only) FIR filters. The table data denotes the average values from processing 15 *miscellaneous* images in USC-SIPI Database [29]. Area reduction estimates are similar to those presented in Section 4.2. The PSNR and SSIM values correspond to the comparison of processed images from the *all-accurate* and *all-approximate* implementations. Average PSNR of up to 52.36 and 47.55 were observed for 2D and 2S modes respectively. Similarly, SSIM of up to 0.99 and 0.97 were observed respectively for the two modes using the approximate multipliers.

The *all-accurate* and *all-approximate* implementations denote the two extremes of the possible multiplier configurations in

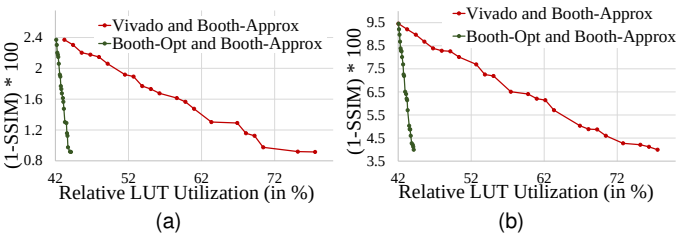


Fig. 14. Pareto-fronts for combinations of accurate and approximate multipliers in an FIR filter with 49 *taps* for processing of two benchmark images— (a) Lena (b) Cameraman. The metrics are relative to that of an all-accurate Vivado multiplier based design

an FIR filter implementation. We have used *Genetic Algorithm (GA)*-based multi-objective design space exploration (DSE) for finding an appropriate combination of multiplier types with a trade-off between resource utilization and processed image quality. An *individual* of the *population* is denoted by a sequence string specifying the type of multiplier used at each position of the FIR filter's hardware implementation. Starting population of 100 individuals, and a maximum of 25 generations have been used for the DSE. Two-point crossover with a probability of 0.5 and a mutation probability of 0.1 are used for *evolution*. We have used two sets of DSE-related experiments. In the first case, the multiplier types are allowed to vary among either Vivado's signed multiplier IP type or our proposed Booth-Approx type. Similarly, the choices in the second experiment are restricted to both the proposed multiplier architectures – Booth-Opt and Booth-Approx. Each possible combination of the choices for the type of multiplier—Vivado's IP-based, Booth-Opt or Booth-Approx—used in the FIR filter's hardware implementation results in a unique point on the design space. Fig. 14(a) and Fig. 14(b) shows the Pareto fronts generated from the two DSE experiments—Vivado + Booth-Approx and Booth-Opt + Booth-Approx for the images used in the experiment. The results correspond to the 2D processing mode with a 7×7-sized convolution window. As evident from the figure, using our proposed multiplier architectures results in a large reduction in resource utilization while still providing an equivalent number of Pareto front design points. Therefore, using our proposed multipliers can result in better application-level optimization.

The extent of application-level improvements depends upon the number of operators in the design that can be approximated. We performed experiments for estimating the effect of the proposed approximate design on ANNs. As a case-study, we implemented the proposed multiplier designs for a single fully-connected layer of an ANN with 85 physical artificial neurons. The resulting design with 8×8 approximate multipliers resulted in significantly lesser resource utilization (6825 $CLBs$ compared to 8285 $CLBs$), critical path delay (3.976 ns compared to 4.423 ns) and power (PDP of 10.88 nJ compared to 12.31 nJ) than one using accurate multipliers. On the other hand, the behavioral analysis of a sample ANN-based inference of MNIST Fashion database showed only 0.04% reduction in accuracy due to the usage of approximate multipliers. Hence, in such applications, the resources saved using the approximate-based design can be used to instantiate more artificial neurons to provide improved parallelism without any significant reduction in inference accuracy. Therefore, the small improvements for a single approximate multiplier can result in significant overall improvements for similar larger applications.

5 Conclusion

In this paper, a design methodology for implementing accurate and approximate signed array-multipliers has been presented. By utilizing the 6-input LUTs and associated carry chains, we presented an area-optimized and energy-efficient implementation of radix-4 Booth's multiplier. We then analyze this implementation and propose an approximate signed multiplier with further improvements in area (LUTs), critical path delay and energy gains. This analysis can be used for implementing more approximate designs with different performance gains. We observe area reductions varying between 47% and 63% across multipliers of different bit-widths. Our proposed designs are also implementable on the newer versions of Xilinx FPGAs, such as Virtex UltraScale+, having 6-input LUTs. We have also provided a high-level application environment for testing accurate/approximate arithmetic functions. We used Gaussian smoothing as a test-case for evaluating the benefits of using the proposed multipliers in an application-scenario and observed minimal degradation with up to 58% reduction in estimated resource utilization.

References

[1] Vivado Design Suite User Guide https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug973-vivado-release-notes-install-license.pdf
[2] I. Kuon et al., "Measuring the Gap Between FPGAs and ASICs," in IEEE TCADICS 2007.
[3] Xilinx 7 Series DSP48E1 Slice, User Guide https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf
[4] Xilinx LogiCORE IP v12.0 https://www.xilinx.com/support/documentation/ip_documentation/mult_gen/v12_0/pg108-mult-gen.pdf
[5] C. S. Wallace, "A Suggestion for a Fast Multiplier," in IEEE Transactions on Electronic Computers, 1964.
[6] L. Dadda, "Some schemes for parallel multipliers," Alta frequenza, vol. 34, no. 5, 1965.
[7] A. D. Booth, "A Signed Binary Multiplication Technique," in The Quarterly Journal of Mechanics and Applied Mathematics 1951.
[8] C. R. Baugh et al.,, "A two's complement parallel array multiplication algorithm," in IEEE TC, vol. 100, no. 12, 1973.
[9] M. Kumm et al., "An efficient softcore multiplier architecture for Xilinx FPGAs," in Computer Arithmetic (ARITH), 2015.
[10] S. Ullah et al., "SMApproxlib: Library of FPGA-based Approximate Multipliers", in DAC, 2018.
[11] S. Ullah et al., "Area-optimized low-latency approximate multipliers for FPGA-based hardware accelerators", in DAC, 2018.
[12] H. Parandeh-Afshar et al., "Measuring and reducing the performance gap between embedded and soft multipliers on FPGAs," in FPL. IEEE, 2011.
[13] S. Mittal, "A Survey of Techniques for Approximate Computing," in ACM Comput. Surv. 48, 4, Article 62 March 2016.
[14] K. et al., "Power-and area efficient Approximate Wallace Tree Multiplier for error-resilient systems," in ISQED 2014.
[15] C. Liu et al., "A low-power, high-performance approximate multiplier with configurable partial error recovery," in DATE 2014.
[16] C.-H. Lin et al., "High accuracy approximate multiplier with error correction," in ICCD 2013.
[17] S. Rehman et al., "Architectural-space exploration of approximate multipliers," in ICCAD 2016.
[18] P. Kulkarni et al., "Trading accuracy for power with an underdesigned multiplier architecture," in VLSI Design 2011.
[19] V. Mrazek et al., "EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in DATE 2017.
[20] I. Qiqieh et al., "Energy-efficient approximate multiplier design using bit significance-driven logic compression," in DATE 2017.
[21] T. Yang et al., "Low-Power and High-Speed Approximate Multiplier Design with a Tree Compressor," in ICCD 2017.
[22] J.-L. Beuchat et al., "Automatic generation of modular multipliers for FPGA applications," in IEEE TC, vol. 57, no. 12, 2008.
[23] A. K. Verma et al., "Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design," in DATE 2008.
[24] Xilinx UltraScale Architecture Configurable Logic Block, User Guide https://www.xilinx.com/support/documentation/user_guides/ug574-ultrascale-clb.pdf
[25] Xilinx 7 Series FPGAs Configurable Logic Block, User Guide https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf
[26] G. W. Bewick, "Fast Multiplication: Algorithms and Implementation," Dissertation, Stanford University, 1994.
[27] Z. Wang et al., "Image Qualifty Assessment: From Error Visibility to Structural Similarity." IEEE TIP, 2004.
[28] W. Liu, et al. Design of Approximate Radix-4 Booth Multipliers for Error-Tolerant Computing, in IEEE TC, Vol. 6, No. 8, 2017.
[29] SIPI Image Database (2019) http://sipi.usc.edu/database/database.php?volume=misc
[30] E. G. Walters, "Partial-product generation and addition for multiplication in FPGAs with 6-input LUTs", In ACSSC 2014.
[31] E. G. Walters, "Array Multipliers for High Throughput in Xilinx FPGAs with 6-Input LUTs", in Computers, MDPI, 2016.
[32] Langhammer, M., Baeckler, G., "High Density and Performance Multiplication for FPGA" in ARITH 2018.
[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in IEEE CVPR 2016.
[34] J. Wang, S. Kuang and S. Liang, "High-Accuracy Fixed-Width Modified Booth Multipliers for Lossy Applications," in TVLSI 2011.
[35] S. -. Jou, Meng-Hung Tsai and Ya-Lan Tsao, "Low-error reduced-width Booth multipliers for DSP applications," in TCAS I 2003.

[36] K. Cho, K. Lee, J. Chung and K. K. Parhi, "Design of low-error fixed-width modified booth multiplier," in TVLSI 2004.
[37] M. A. Song, L. D. Van, and S. Y. Kuo, "Adaptive low-error fixedwidth Booth multipliers," IEICE Trans. Fundamentals, 2007.
[38] H. Ko and S. Hsiao, "Design and Application of Faithfully Rounded and Truncated Multipliers With Combined Deletion, Reduction, Truncation, and Rounding," in TCAS II: Express Briefs 2011.