IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 25, 2020, accepted March 31, 2020, date of publication April 3, 2020, date of current version April 16, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2985345

On the Design of Logarithmic Multiplier Using Radix-4 Booth Encoding

RATKO PILIPOVIĆ, (Member, IEEE), AND PATRICIO BULIĆ, (Member, IEEE)
Faculty of Computer and Information Science, University of Ljubljana, 1000 Ljubljana, Slovenia

Corresponding author: Ratko Pilipović (ratko.pilipovic@fri.uni-lj.si)

This work was supported in part by the Slovenian Research Agency (ARRS) under Grant P2-0359 (National Research Program Pervasive Computing) and Grant P2-0257 (System on Chip with Integrated Optical, Magnetic and Electrochemical Sensors), and in part by the Slovenian Research Agency (ARRS) and Ministry of Civil Affairs, Bosnia and Herzegovina, under Grant BI-BA/19-20-047 (Bilateral Collaboration Project).

ABSTRACT This paper proposes an energy-efficient approximate multiplier which combines radix-4 Booth encoding and logarithmic product approximation. Additionally, a datapath pruning technique is proposed and studied to reduce the hardware complexity of the multiplier. Various experiments were conducted to evaluate the multiplier's error performance and efficiency in terms of energy and area utilization. The reported results are based on simulations using TSMC-180nm. Also, the applicability of the proposed multiplier is examined in image sharpening and convolutional neural networks. The applicability assessment shows that the proposed multiplier can replace an exact multiplier and deliver up to a 75% reduction in energy consumption and up to a 50% reduction in area utilization. Comparative analysis with the state-of-the-art multipliers indicates the potential of the proposed approach as a novel design strategy for approximate multipliers. When compared to the state-of-the-art approximate non-logarithmic multipliers, the proposed multiplier offers smaller energy consumption with the same level of applicability in image processing and classification applications. On the other hand, some state-of-the-art approximate logarithmic multipliers exhibit lower energy consumption than the proposed multiplier but deliver significant performance degradation for the selected application cases.

INDEX TERMS Approximate computing, arithmetic circuit design, booth encoding, logarithmic multipliers, multipliers, power-efficient processing, truncated multipliers.

I. INTRODUCTION

As the data processing in error-tolerant applications becomes more and more power demanding, approximate computing surfaced as the solution to obtain significant gains in computational throughput and to achieve more power-efficient processing [1]. Approximate computing assumes small approximations in computing while maintaining an acceptable quality of results [1]–[5]. Approximate computing has become a popular strategy for arithmetic circuit design, where energy-efficient design is more important than accuracy, and the challenge is to achieve the best trade-off between accuracy and design efficiency [3], [4], [6]. The principal requirement for an approximate arithmetic circuit is to exhibit a controllable error with design efficiency as the central goal. The multipliers are the indispensable components in many systems, e.g., digital signal processors, embedded vision systems and hardware accelerators. At the same time, multipliers represent the most complex arithmetic components in the digital implementation of these systems (e.g., hardware deep neural networks) [7]–[9]. Thus, obtaining an efficient multiplier design is of the highest importance. The multiplier design can be remarkably simplified if we exclude the requirement for exact computation, so various versions of approximate multiplier design have been proposed recently.

Two design approaches dominate in the field of approximate multipliers: approximate logarithmic and approximate non-logarithmic. An approximate logarithmic multiplier employs binary logarithms to simplify multiplication. In the logarithm domain, multiplication is replaced with addition. Approximate logarithmic multipliers employ error correction circuits to suppress a significant error in computation. Unfortunately, the employment of error correction circuits increases the overall complexity of such multipliers. Approximate non-logarithmic multipliers rely on the simplification of two stages: the partial product addition

The associate editor coordinating the review of this manuscript and approving it for publication was Sedat Akleylek.

stage and the partial product generation stage. The partial product addition stage employs compressors [10]–[12], which transform multi-operand addition into a two-operand addition. The simplification of the partial product generation stage is more valuable - for an efficient multiplier design, the partial product generation stage should produce as little partial products as possible. For this purpose, approximate non-logarithmic multipliers employ Booth encoding [13] that can decrease the number of partial products to n/r, where n represents the number of bits in inputs, and r denotes the exponent of the utilized Booth encoding radix. The usage of Booth encoding is usually limited to the radix-4 encoding due to complex logic required to encode higher radix multiples. On the other hand, high radix Booth encoding can be achieved with approximate computing by allowing a small computational error in the generation of hard multiplies.

The approximate non-logarithmic multiplier strategy brings better accuracy and higher design complexity compared to approximate logarithmic multipliers. On the other hand, the approximate logarithmic multiplier strategy sets the design efficiency as the primary goal, while accuracy comes second. In this study, we propose a multiplier design, which relies on both strategies. We anticipate that in error-resilient applications (e.g., convolutional neural networks), the proposed strategy would offer the same performance as approximate non-logarithmic multipliers but would exhibit lower energy consumption. The findings in this study reveal the potential of proposed strategy for designing state-of-the-art approximate multipliers aimed for convolutional neural networks as well as for a broader specter of error-resilient applications.

The contributions of this paper can be summarized as follows:

1) In this paper, we combine radix-4 Booth encoding and logarithmic product approximation to create a new generation of approximate multipliers that achieve a good trade-off between accuracy and design efficiency. The proposed design strategy utilizes an approximate logarithmic multiplier to generate and approximate high-radix partial products.
2) A datapath pruning technique is introduced and used to simplify the high-radix logarithmic partial product generation stage while maintaining a small. Furthermore, the proposed datapath pruning technique can be used in any approximate logarithmic multiplier.
3) A highly adjustable design with three parameters, which are introduced to tune the trade-off between circuit complexity and approximation error, is presented.

The remainder of this paper is organized as follows. Section II reviews the related work in the field of approximate multipliers. Section III describes the architecture of the proposed multiplier. Section IV deals with the error characteristics of the proposed multiplier. The synthesis results are presented in Section V, and the experiments on the image sharpening and convolutional neural networks are discussed in Section VI. Lastly, the conclusion of the paper is presented in Section VII.

II. RELATED WORK

We group the related work into two approaches based on two paradigms of approximate multiplier design: the approximate logarithmic and approximate non-logarithmic multipliers.

A. APPROXIMATE LOGARITHMIC MULTIPLIERS

Mitchell's multiplier [14] represents the first approximate logarithmic multiplier, where the binary logarithm is used to approximate multiplication. The main drawback of Mitchell's multiplier is low accuracy, which can be suppressed by an error correction circuit. With the error correction circuit, Mitchell's multiplier exhibits a mean relative error of 3.8% and the maximal relative error of 11%. The complexity of the error correction circuit represents a significant design drawback. Mahalingam and Ranganathan [15] proposed an improvement to Mitchell's multiplier through operand division. The proposed design increases the accuracy of Mitchell's multiplier by 44.7%, on average, but almost doubles the number of logic gates. To reduce area and energy consumption of Mitchell multiplier, Gandhi *et al.* [16] employed approximate leading-one detectors. With the employment of approximate leading-one detectors, authors lowered the energy consumption of Mitchell multiplier by 55 %. On the other hand, the developed multiplier exhibits significant error for small numbers.

Kim *et al.* [17] employed the truncation of operands to achieve a more efficient Mitchell's multiplier design. The authors truncate the logarithmic representation of the multiplier's input to utilize smaller barrel shifters. The proposed truncation significantly decreases delay, power, and area compared to the original Mitchell's multiplier, but with the cost of a higher error. Recently, Kim *et al.* [18] have addressed the problem of high error in truncated Mitchell multiplier by proposing iterative truncated Mitchell multiplier. The iterative approach significantly increases accuracy with the price of increased power consumption.

Liu *et al.* [19] utilize truncated approximate adders for mantissa addition, which delivers a design with smaller barrel shifters. Their multiplier offers similar accuracy as Mitchell's multiplier and at the same time, delivers a smaller power-delay product. The truncation is also used in [20], where the authors introduced the Dynamic Range Unbiased Multiplier (DRUM). DRUM delivers good results in area reduction but keeps a small error only for specific input combinations.

To improve the accuracy of Mitchell's multiplier, Babić *et al.* developed the iterative logarithm multiplier (ILM) [21]. ILM is an efficient and straightforward multiplier that achieves arbitrary accuracy through an iterative procedure. Unlike Mitchell's multiplier, ILM employs a simpler product approximation. Because of the simpler product approximation, basic ILM has a relatively high mean relative error of around 10%. When the procedure is iteratively

applied, the accuracy of the multiplier increases. With only one iteration for error correction, the ILM design delivers a mean relative error of around 1%. The improvement in accuracy increases area utilization. The desirable feature of ILM is the capability to perform each iteration concurrently in a pipelined fashion. This feature made the ILM multiplier especially useful in hardware neural networks [22].

Ahmed and Srinivas [23] addressed the problem of complexity in ILM by developing the truncated iterative multiplier (TIM). TIM follows the ILM strategy of product approximation, but unlike ILM, TIM truncates the terms in product approximation to get smaller adders and shifters. Compared to ILM, truncation results in a significant decrease in the chip area but fails to deliver improvement in delay. From an accuracy point of view, TIM achieves a smaller error compared to ILM.

Recently, number rounding has become a popular approach for approximate logarithmic multiplier design. Zendagani *et al.* [24] developed the rounding based approximate multiplier (ROBA), where the input operands are rounded to the nearest exponent of two. Three hardware implementations were proposed: unsigned U-ROBA, signed S-ROBA, and approximate signed AS-ROBA. The proposed designs exhibit a mean relative error of around 3%. The authors managed to reduce the delay and energy compared to the DRUM multiplier. The design by Zendagani *et al.* [24] has been improved by Vahdat *et al.* [25], where the authors have combined truncation and rounding to deliver an efficient approximate multiplier design with significant reductions in area, energy consumption, and delay compared to the exact multiplier. On the other hand, the proposed approach exhibits significant mean relative error that can go up to 11%. From an accuracy point of view, TIM achieves a smaller error compared to ILM. Finally, Ansari *et al.* [26] replaced the exact adders with the energy-efficient inexact adders in the rounding-based multiplier. Usage of inexact adders brought up to 40% lower energy consumption compared to conventional Mitchell multiplier.

B. APPROXIMATE NON-LOGARITHMIC MULTIPLIERS

A considerable attention is devoted to intermediate partial product addition, more specifically, the design of approximate compressors. Momeni *et al.* [27] derived an inexact compressor by modifying the truth table of an exact compressor. Similarly to [27], Reddy *et al.* [28] altered the truth table of exact 4:2 compressor to obtain the energy-efficient inexact 4:2 compressor. The proposed compressor has a lower error than the compressor in [27]. Although the inexact compressor offers chip area and power savings, it lacks in accuracy tuning. Lin and Lin *et al.* [29] overcame this problem by introducing an inexact compressor with partial error correction. Their design is characterized by a smaller error, but the chip area increased compared to [27].

Ha and Lee *et al.* [30] proposed an inexact compressor with error correction by further simplifying the design proposed by Momeni *et al.* [27]. This design outperforms both Momeni *et al.*'s and Lin *et al.*'s designs in area utilization and delay but exhibits a larger error. Jiang *et al.* [31] developed an inexact adder to simplify the partial product generation stage and proposed an error correction circuit to suppress the error. Venkatachalam and Ko *et al.* [32] proposed the partial product transformation that allows simple and highly accurate partial tree reduction with inexact compressors. The proposed transformation lowers the probability of bits in the partial product tree being equal to one. The proposed approximation is utilized in two variants of 16-bit multipliers. The authors reported that two proposed multipliers achieve power savings of 72% and 38%, respectively, compared to an exact multiplier. Qiqieh *et al.* [33] presented an algorithm for compressing the partial product tree which relies on bit-significance. When applied to the radix-4 128-bit multiplier, proposed compression delivers reductions of 65% in critical delay, and almost 45% in area utilization with only 0.003% mean relative error.

Esposito *et al.* [34] developed XOR-free inexact compressors that achieve a small error. The XOR-free compressors deliver a significant reduction in area utilization and power consumption. Furthermore, the authors proposed an algorithm for the placement of XOR-free compressors into the partial product tree. Ahmadinejad *et al.* [35] designed inexact compressors in FinFET technology. Similarly to existing approaches, authors altered truth tables for exact 4:2 and 5:2 compressors to create inexact compressors. The proposed 4:2 and 5:2 compressors improve the power delay product (PDP), on average by 59% and 68%, and area by 60% and 75%, respectively. Sabetzadeh *et al.* [36] adopted majority logic approximation to develop inexact compressors in FinFET technology. Compared to [35], the proposed compressor is designed with a smaller number of transistors and exhibits a lower error. The authors report that this compressor is superior to its previous counterparts in terms of delay, power consumption, power delay product (PDP) and area, and improves these parameters on average by 32%, 68%, 78%, and 66%, respectively.

Various approaches try to simplify the Booth encoding stage. Jiang *et al.* [37] proposed the use of an inexact recoding adder for more efficient radix-8 Booth encoding. The proposed recoding adder accurately adds the higher half of input bits while the lower half bits are added approximately. Waris *et al.* [38] presented hybrid low-radix encoding, which combines approximate radix-8 Booth encoding and exact radix-4 Booth encoding. The radix-4 Booth encoding is applied to the most significant bits, while the novel approximate radix-8 encoding is applied to the least significant bits. Compared to [37], the hybrid low-radix encoding exhibits a smaller error and delivers lower energy consumption. Similarly, Liu *et al.* developed a simpler radix-4 Booth encoding stage [39]. Compared to the exact version, the proposed stage utilizes simpler logic for partial product generation. The introduced error is negligible, but on the other hand, the design does not reduce the number of partial products compared to the exact

radix-4 Booth encoding. Ansari *et al.* [40] proposed employment of approximate radix-4 Booth encoding and 4-2 compressors to obtain low-power multiplier. Combination of radix-4 Booth encoding and 4-2 compressors delivered the 44% smaller power-delay product than previous approximate radix-4 Booth multipliers.

Zervakis *et al.* [41] introduced a technique for partial product perforation in the Booth encoding. It leads to a simpler partial product addition stage but also a significant decrease in the accuracy. To address the problem of accuracy, Leon *et al.* [42] improved the idea of partial product perforation by introducing the hybrid radix Booth encoding technique. Although the proposed design delivers a small mean relative error, the occurrence of significant errors is not negligible. The same authors extended the idea of partial product perforation with the rounding concept [43]. While the perforation concept is applied to the multiplicand, the rounding concept is applied to the multiplier. The aim is to simplify the generation of each partial product. Recently, Liu *et al.* [44] proposed the usage of approximate redundant binary multipliers. The authors simplified the partial product generation stage through modification of the Karnaugh table, and proposed the use of approximate redundant compressors.

III. LOBO MULTIPLIER

We propose a logarithmic-booth encoding (LOBO) multiplier that uses the radix-4 Booth encoding to compute the higher part of the product and the logarithmic approximation to approximate the lower (least significant) part of the product. In this section, we give a detailed description of the LOBO design and its adaption to the 16-bit signed multiplier.

A. MULTIPLICAND'S ENCODING

Let us write an n-bit signed binary number as:

$$b_{n-1}b_{n-2}\dots b_d b_{d-1}b_{d-2}\dots b_1 b_0,$$

where n is an even integer. Let the above binary number be coded in two's complement. Then, its decimal value is

$$X = -b_{n-1}\cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i.$$

Let the bits in the above binary number be divided into two parts: the $(n-d)$-bit most-significant (MS) part and the d-bit least-significant (LS) part. We encode the MS part with radix-4 encoding [13], while for the LS part, we use binary encoding:

$$X = \sum_{j=1}^{(n-d)/2} \hat{b}_j^{R4}\cdot 4^{(j-1)+d/2} + B_0 \quad , \tag{1}$$

where

$$\hat{b}_j^{R4} = -2b_{2j+1} + b_{2j} + b_{2j-1}, \quad \hat{b}_j^{R4} \in \{0, \pm 1, \pm 2\}, \tag{2}$$

and

$$B_0 = -b_{d-1}\cdot 2^{d-1} + \sum_{i=0}^{d-2} b_i \cdot 2^i. \tag{3}$$

The radix-4 encoding of $\hat{b}_1^{R4}$ implies that B_0 is a d-bit signed number and requires that the length of the MS part is an even number, so d is also an even number. The encoding of a 16-bit number is

$$\overbrace{b_{15}b_{14}\underbrace{b_{13}b_{12}}_{\hat{b}_2^{R4}}}^{\hat{b}_3^{R4}}\overbrace{\underbrace{b_{11}b_{10}}_{\hat{b}_1^{R4}}\underbrace{b_9 b_8 b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0}_{B_0}}, \tag{4}$$

and

$$\overbrace{b_{15}b_{14}\underbrace{b_{13}b_{12}}_{\hat{b}_1^{R4}}}^{\hat{b}_2^{R4}}\overbrace{b_{11}b_{10}b_9 b_8 b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0}^{B_0}, \tag{5}$$

for $d = 10$ and $d = 12$, respectively.

The main idea behind the proposed LOBO multiplier is to generate few most-significant partial products using the radix-4 Booth encoding and to approximate the remaining high-radix product, that evolves from B_0, with an approximate logarithmic multiplier. The selection of a value for d shapes the overall performance of the LOBO multiplier. If d is small, LOBO exhibits good accuracy but produces a large number of radix-4 partial products. The opposite case brings fewer radix-4 partial products at the expense of accuracy. From the design perspective, the choice of d determines the complexity of the partial product addition stage.

B. PRODUCT APPROXIMATION

Let X and Y be two n-bit two's complement numbers and let the number X be encoded using the proposed encoding as in Eq. (1). Given the previously described proposed encoding we can derive a mathematical formula for the product $X \cdot Y$ as:

$$X\cdot Y = \sum_{j=1}^{(n-d)/2} PP_j \cdot 4^{(j-1)+d/2} + B_0 \cdot Y, \tag{6}$$

where $PP_j = \hat{b}_j^{R4}\cdot Y$. The partial products PP_j are easy to generate without multiplication due to the use of radix-4 Booth encoding. The only multiplication in Eq. (6) is required by the term $B_0 \cdot Y$. As B_0 is a signed integer number, we can write

$$B_0 = s(B_0)\cdot(2^{k_{|B_0|}} + B_{00}), \tag{7}$$

where $s(\cdot)$ denotes the sign operation, and

$$B_{00} = |B_0| - 2^{k_{|B_0|}}, \tag{8}$$

where $|B_0|$ is the absolute value of B_0 and $k_{|B_0|} = \lfloor \log_2 |B_0| \rfloor$ represents the position of the leading one bit in $|B_0|$. Using Eq. (7) we can express the product $B_0 \cdot Y$ as:

$$B_0\cdot Y = s(B_0)\cdot Y\cdot 2^{k_{|B_0|}} + s(B_0)\cdot B_{00}\cdot Y. \tag{9}$$

Similarly to B_0, we can express Y as:

$$Y = s(Y)\cdot(2^{k_{|Y|}} + Y_{00}), \tag{10}$$

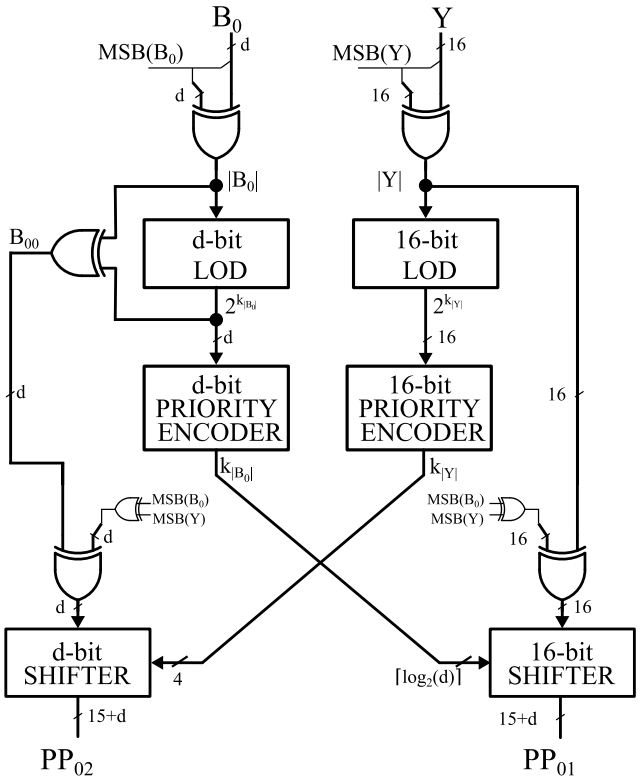


FIGURE 1. The logarithmic partial product generation (LPPG) stage for the 16-bit LOBO multiplier.

where $Y_{00} = |Y| - 2^{k_{|Y|}}$, $|Y|$ represents the absolute value of Y and $k_{|Y|} = \lfloor \log_2 |Y| \rfloor$ denotes the position of the leading one bit in $|Y|$. By combining Eq. (9) and Eq. (10) we can write

$$B_0 \cdot Y = PP_{01} + PP_{02} + s(B_0) \cdot s(Y) \cdot B_{00} \cdot Y_{00}, \quad (11)$$

where

$$PP_{01} = s(B_0) \cdot s(Y) \cdot |Y| \cdot 2^{k_{|B_0|}}, \quad (12)$$

and

$$PP_{02} = s(B_0) \cdot s(Y) \cdot B_{00} \cdot 2^{k_{|Y|}}. \quad (13)$$

The terms PP_{01} and PP_{01} require only the shift operations while the term $s(B_0) \cdot s(Y) \cdot B_{00} \cdot Y_{00}$ requires multiplication. According to [21], we may neglect this term and approximate $B_0 \cdot Y$ as:

$$B_0 \cdot Y \approx PP_{01} + PP_{02}. \quad (14)$$

The product approximation in the proposed LOBO multiplier is:

$$X \cdot Y \approx \sum_{j=1}^{(n-d)/2} PP_j \cdot 4^{(j-1)+d/2} + PP_{01} + PP_{02} \quad (15)$$

To obtain PP_{01} and PP_{02}, we employ the logarithmic partial product generation stage (LPPG), shown in Fig. 1. The LPPG stage works as follows. At its input, we employ two XOR logic gates to approximate the absolute values $|Y|$ and $|B_0|$, respectively. This way, we replace the two's complement with one's complement. This approximation indeed introduces an additional error in the computation of product $B_0 \cdot Y$, but it delivers a more straightforward hardware design. Then, the absolute values $|Y|$ and $|B_0|$ enter the leading-one detectors (LOD) to obtain the leading-one bits $2^{k_{|B_0|}}$ and $2^{k_{|Y|}}$, respectively. As $2^{k_{|B_0|}}$ represents the leading-one bit in $|B_0|$, we employ XOR gates to remove the leading-one bit from $|B_0|$, thus retrieving B_{00}. The leading-one bits $2^{k_{|B_0|}}$ and $2^{k_{|Y|}}$ enter also the priority encoders to generate $k_{|B_0|}$ and $k_{|Y|}$, respectively. These two values are later used in the barrel shifters to indicate the number of shifts. We implemented the priority encoder module with one layer of logic-OR gates because its input is in a one-hot format. The terms $s(B_0)s(Y)|Y|$ and $s(B_0)s(Y)B_{00}$ are approximated by employing three XOR gates. Firstly, a XOR gate is used to generate the product $s(B_0)s(Y)$. Secondly, two XOR gates are used to generate the terms $s(B_0)s(Y)|Y|$ and $s(B_0)s(Y)B_{00}$. Again, we approximate two's complement with one's complement. Finally, two barrel shifters are used to produce the partial products PP_{01} and PP_{02}.

C. DATAPATH PRUNING

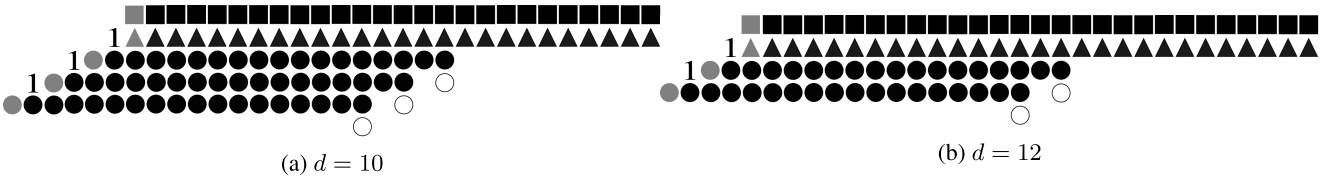
The most complex components in the LPPG stage from Fig. 1 are the LOD detectors, priority encoders, and barrel shifters. They all introduce a significant delay. To reduce the area and minimize the delay of these components, we propose a technique we call datapath pruning. Datapath pruning approximates the shift operations and reduces the number of shifts in the barrel shifters. Consequently, it also simplifies the encoders and LODs. We differ two types of datapath pruning: soft datapath pruning and hard datapath pruning. The basic idea of soft datapath pruning is to ignore a shifter when the number of shifts is small. In contrast to soft datapath pruning, which ignores a shifter, hard datapath pruning sets the output of a shifter to zero when the number of shifts is small. Hard datapath pruning induces a more significant approximation error but offers a more efficient circuit design of the LPPG stage. On the other hand, soft datapath pruning delivers a smaller approximation error but requires a more complex circuitry. The proposed soft and hard datapath pruning are as follows.

1) SOFT DATAPATH PRUNING

Here, the partial product PP_{01} is approximated as:

$$PP_{01} = \begin{cases} s(B_0)s(Y)|Y|, & k_{|B_0|} < T_S \\ s(B_0)s(Y)|Y|2^{k_{|B_0|}}, & k_{|B_0|} \geq T_S, \end{cases} \quad (16)$$

where T_S is an integer threshold. This means that for small $k_{|B_0|}$ PP_{01} is equal to $s(B_0) \cdot Y$ (without shift). Otherwise, when $k_{|B_0|} \geq T_S$ the term $s(B_0) \cdot Y$ is shifted left $k_{|B_0|} - T_S$ times and T_S zeroes are appended to its right. This way we need a smaller barrel shifter. To find $k_{|B_0|}$, one should firstly use a leading one detector and then a binary encoder. Eq. 16 shows that we are interested in the position of the leading one



(a) $d = 10$ (b) $d = 12$

FIGURE 2. The partial product tree for (a) $d = 10$, and (b) $d = 12$. Legend: ● the partial product bits from the exact radix-4 encoding; ■: the PP_{01} bits; ▲: the PP_{02} bits; ●, ■, and ▲,: the inverted MSB bits of the partial products; ○: the sign bits from the exact radix-4 partial products.

bit only if it is at the place higher or equal to T_S. Consequently, we should detect the leading one bit in the following word:

$$b_{d-1}, b_{d-2}, \ldots, b_{T_S}, \tag{17}$$

where b_i represents the i-th bit of $|B_0|$. Besides a smaller barrel shifter, the LOD detector and priority encoder are also smaller when the soft datapath pruning is applied.

2) HARD DATAPATH PRUNING

Here, the partial product PP_{02} is approximated as:

$$PP_{02} = \begin{cases} 0, & k_{|Y|} < T_H \\ s(B_0)s(Y)B_{00}2^{k_{|Y|}}, & k_{|Y|} \geq T_H, \end{cases} \tag{18}$$

where T_H is an integer threshold. With this approach we discard PP_{02} for small values of $k_{|Y|}$. The error introduced by the hard datapath pruning is larger than the error introduced by the soft datapath pruning. Again, the leading-one detector is applied only to the upper $16 - T_H$ bits in $|Y|$.

Here we would like to emphasize that it is also possible to choose different combinations of soft and hard datapath pruning. To determine the optimal choice of datapath pruning, we implemented four versions of the LOBO multiplier: hard datapath pruning for both PP_{01} and PP_{02}; soft datapath pruning for both PP_{01} and PP_{02}; soft datapath pruning for PP_{01} and hard datapath pruning for PP_{02} (the proposed solution in this paper); hard datapath pruning for PP_{01} and soft datapath pruning for PP_{02}. For each version of the LOBO multiplier, we obtained area, power-delay-product (PDP), and classification accuracy of Cuda-convent neural networks when these multipliers are employed in inference. Table 1 shows the results obtained where the multipliers' area and PDP are normalized. We can see that the proposed solution (soft datapath pruning for PP_{01} and hard datapath pruning for PP_{02}) brings a minimal increase in area (3%) and PDP (1%) with hardly any degradation in the classification accuracy. Therefore, in the text that follows, only LOBO with soft datapath pruning for PP_{01} and hard datapath pruning for PP_{02} is considered. In Section IV, we assess in more detail the impact of the selected datapath pruning on accuracy.

The thresholds T_S and T_H have a significant impact on the performance of the LOBO multiplier. If they are small, then the LOBO multiplier delivers a proper error distribution but requires a more complex LPPG stage in terms of area, delay, and power. On the other hand, when they are large, LOBO utilizes a smaller LPPG stage in terms of area, delay, and power, but has a significant error for small numbers.

TABLE 1. Datapath pruning and its influence on area, power-delay-product (PDP), and classification accuracy for LOBO with $d = 12$, $T_H = 12$, and $T_S = 8$.

Datapath pruning	Area	PDP	Classification accuracy
Hard for PP_01 and PP_02	1.00	1.00	0.782+/-0.008
Soft for PP_01 and PP_02	1.14	1.27	0.801+/-0.007
Soft for PP_01, hard for PP_02	1.03	1.01	0.799+/-0.007
Hard for PP_01, soft for PP_02	1.09	1.15	0.800+/-0.006

In Section IV, we give a detailed analysis of the impact of the thresholds on the error and the complexity of the multiplier's design. To differentiate between specific LOBO multipliers in the rest of the paper, we will use the notation *LOBOd* or more detailed *LOBOd* − -T_H/T_S. For example, *LOBO*12 − -12/8 represents the LOBO multiplier with $d = 12$, $T_H = 12$, and $T_S = 8$.

D. PARTIAL PRODUCT ADDITION STAGE

In the partial product addition stage we employ the Wallace tree [45] to reduce the number of partial products to two, whereas final partial product addition is implemented using a prefix (fast) adder [13]. Fig. 2a and Fig. 2b present the partial product tree for *LOBO*10 and *LOBO*12, respectively. *LOBO*10 produces five partial products whereas when *LOBO*12 produces four partial products.

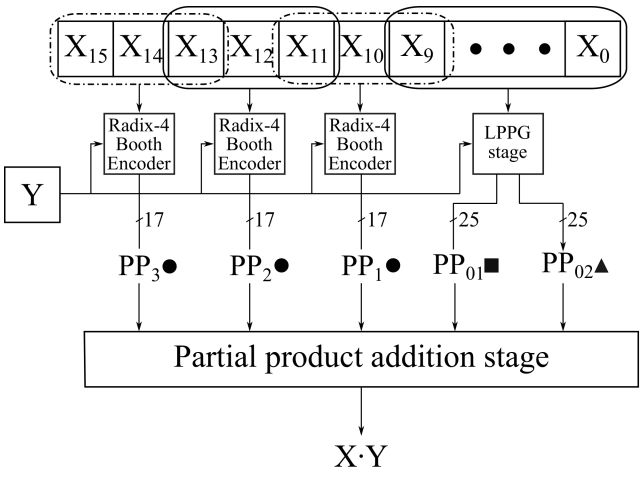


FIGURE 3. The architecture of *LOBO*10.

The overall architecture of the LOBO multiplier, presented in Fig. 3, consists of radix-4 partial product generation, the LPPG stage, and the partial product addition stage.

IV. ERROR CHARACTERISTICS OF THE LOBO MULTIPLIER

The LOBO multiplier is characterized with three parameters: d, T_S and T_H. The parameter d denotes the length of the least significant part (Eq. (1), Eq. (4), Eq. (5)) of the multiplicand, which is approximated in the LPPG stage, while the parameters T_S and T_H represent the pruning thresholds (Eq. (16) and Eq. (18)). In this section, we present the error characteristics of the LOBO multiplier. First, we examine how the mean relative error (MRE) depends on these parameters. Secondly, we asses the error distribution for selected combinations of these parameters. This evaluation provides a more in-depth insight into the error induced by the approximation. All analyses presume a uniform distribution of input numbers and are carried out for every combination of input operands for a 16-bit signed multiplier.

A. INFLUENCE OF PARAMETERS ON ERROR

We asses the influence of the parameters d, T_H, and T_S on MRE that is obtained for every possible combination of input numbers. Firstly, MRE was assessed for $d \in \{6, 8, 10, 12, 14\}$. It can be observed from Fig. 4 that MRE increases exponentially with d, which is in accordance with Eq. (11).

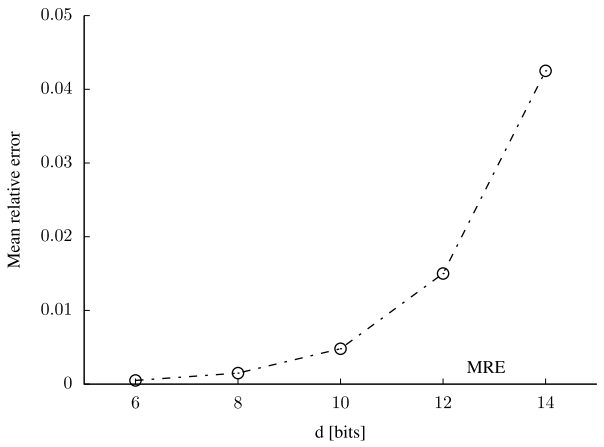


FIGURE 4. The influence of d on MRE.

Secondly, we asses the influence of datapath pruning parameters T_H and T_S on MRE. We examine this influence only for $d \in \{10, 12\}$ as these values provide hardware efficient designs while keeping a small MRE. The only values for T_H that result in a significant reduction in hardware are 8 and 12. Similarly, the only viable values for T_S are 6 and 8. For this reason, in the rest of this work, we concentrate only on $T_H \in \{0, 8, 12\}$ and $T_S \in \{0, 6, 8\}$. Here, the value 0 denotes that there is no datapath pruning. MRE for $T_H \in \{0, 8, 12\}$, and $T_S \in \{0, 6, 8\}$ are presented in Tab. 2. As can be observed, the pruning parameters have a lower impact on the decrease of accuracy than parameter d. But, as shown in Section III, T_H and T_S have a large impact on the hardware complexity of the LPPG stage because they decrease the size of the barrel shifters. The smaller size, in turn, has a positive impact on power, area, and delay at the expense of a relatively small decrease in the accuracy.

TABLE 2. The influence of datapath pruning parameters T_H and T_S on MRE.

		$T_H = 0$	$T_H = 8$	$T_H = 12$
$d = 10$	$T_S = 0$	0.49	0.50	0.62
	$T_S = 6$	0.53	0.54	0.68
$d = 12$	$T_S = 0$	1.50	1.53	1.93
	$T_S = 8$	1.69	1.72	2.17

B. RELATIVE ERROR DISTRIBUTION AND ITS RATE

To further characterize the error performance of LOBO, we evaluate the relative error rate and its distribution, i.e., the probability that relative error is smaller than a specific value.

TABLE 3. The probability that relative error is smaller than a specific value for the 16-bit LOBO multiplier.

Multiplier	<5 [%]	<10 [%]	<15 [%]	<20 [%]	<25 [%]
$LOBO10$–$0/0$	98.27	99.13	99.57	99.88	100.00
$LOBO10$–$8/6$	98.18	99.03	99.45	99.74	99.87
$LOBO10$–$12/6$	97.76	98.82	99.28	99.58	99.72
$LOBO12$–$0/0$	93.02	96.48	98.27	99.51	100.00
$LOBO12$–$8/8$	92.70	96.07	97.76	98.94	99.45
$LOBO12$–$12/8$	90.99	95.23	97.08	98.39	98.85

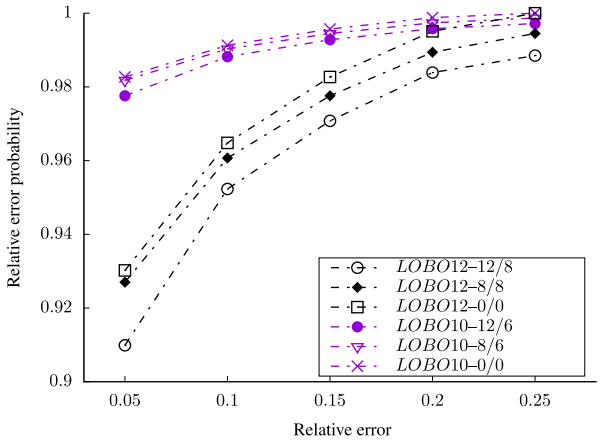


FIGURE 5. The probability that relative error is smaller than a specific value for the 16-bit LOBO multiplier.

The relative error distribution is presented in Table 3 and in Fig. 5. The results show that parameter d has a significant impact on error distribution. For example, the number of outputs whose relative error is below 5% significantly decreases (from 98% to 93%) when the parameter d increases from 10 to 12. The impact of T_H and T_S on the relative error distribution is almost negligible. One could say that the choice of encoding (d) represents the coarse-grained approximation, while the choice of datapath pruning (T_H and T_S) represents the fine-grained product approximation.

TABLE 4. Mean relative error rate for different ranges of input numbers.

Multiplier	$[0, 2^8)$	$[2^8, 2^9)$	$[2^9, 2^{12})$	$[2^{12}, 2^{14})$	$[2^{14}, 2^{15})$
$LOBO10$–$0/0$	8.77 %	9.33 %	1.58 %	0.29 %	0.11 %
$LOBO10$–$8/6$	45.05 %	9.38 %	1.62 %	0.30 %	0.11 %
$LOBO10$–$12/6$	45.05 %	30.43 %	5.28 %	0.30 %	0.11 %
$LOBO12$–$0/0$	8.77 %	9.33 %	6.37 %	1.16 %	0.44 %
$LOBO12$–$8/8$	96.09 %	9.44 %	6.41 %	1.20 %	0.45 %
$LOBO12$–$12/8$	96.09 %	30.62 %	20.91 %	1.20 %	0.45 %

Finally, Table 4 presents the LOBO's mean relative error for different ranges of input numbers. Here, we evaluate the dependence of MRE on different values of input operands. As can be observed, the proposed LOBO multipliers have a significant mean relative error for small input values and vice versa. The large MRE for small input values is a consequence of the logarithmic approximation of the d-bit multiplicand's LS part and datapath pruning (T_H and T_S) used in the LOBO multiplier. Both affect the accuracy of the overall product. Nevertheless, in Section VI, we show that the significant relative error for small numbers does not affect the performance of some real-world error-resilient applications.

TABLE 5. The synthesis results. The LOBO multipliers follow the notation *LOBOd* − -T_H/T_S, where d, T_H, and T_S denote the design parameters.

Multiplier	Delay (ns)	Power (mW)	Area (μm^2)	PDP ($ns \times mW$)	NMED ($\cdot 10^{-4}$)
$Exact\ radix$–4	18.43	6.69	26505	123.35	-
$LOBO10$–$0/0$	19.14	4.40	20387	84.22	4.30
$LOBO10$–$8/6$	16.79	3.90	17527	65.55	4.40
$LOBO10$–$12/6$	15.86	3.51	16551	55.66	4.60
$LOBO12$–$0/0$	25.14	3.15	17386	60.28	17.00
$LOBO12$–$8/8$	16.69	2.62	14024	41.08	17.89
$LOBO12$–$12/8$	14.61	2.15	12126	31.45	18.45
$Mitchell$ [14]	18.48	2.73	12845	50.44	92.70
$Mitchell$–$trunc6$ [17]	12.99	1.51	7981	19.61	144.30
$Mitchell$–$trunc8$ [17]	16.11	1.76	9113	28.67	105.6
ALM–$SOA10$ [19]	15.71	1.45	8696	22.85	83.47
ALM–$SOA11$ [19]	13.98	1.33	8213	18.59	80.55
$DRUM6$ [20]	22.88	2.87	14868	65.60	34.00
S–$ROBA$ [24]	24.40	2.73	19336	66.76	69.01
AS–$ROBA$ [24]	20.38	2.71	16615	55.30	69.44
ILM–AA [26]	12.36	1.41	6893	17.43	72.00
HLR–$BM2$ [38]	15.98	3.52	15103	56.25	0.07
$R4ABM1$–14 [39]	14.76	6.58	25138	97.00	0.09
$R4ABM1$–16 [39]	13.59	6.73	24288	91.46	0.31
$R4ABM2$–14 [39]	14.57	6.32	24724	92.08	0.06
$R4ABM2$–16 [39]	14.46	6.21	24498	89.80	0.30
$RAD1024$ [42]	14.54	3.09	13337	44.92	44.35

V. SYNTHESIS RESULTS

In this section, we evaluate and compare the hardware performance in terms of power, area, delay, and power-delay-product (PDP) of six LOBO multipliers, an exact radix-4 multiplier, and several state-of-the-art approximate multipliers. In addition to hardware performance, we compare the normalized mean error distance (NMED) for all evaluated multipliers. The NMED is defined in [19], [38], [39] as average error distance over all input combinations normalized by the maximum output of the exact multiplier.

The following state-of-the-art approximate non-logarithmic multipliers were chosen for comparison: *RAD*1024 [42], *R4ABM*1 − -14 [39], *R4ABM*1 − -16 [39], *R4ABM*2 − -14 [39], *R4ABM*2 − -16 [39], and *HLR* − -*BM*2 [38]. The approximate multipliers *HLR* − -*BM*2, *R4ABM*s, and *RAD*1024 are implemented to accumulate the partial products accurately with a Wallace tree, just as the proposed LOBO multipliers. We have not included the approximate redundant binary multipliers [44] in the experiments, as their results reveal that these multipliers have higher power consumption compared to state-of-the-art approximate multipliers. The following state-of-the-art approximate logarithmic multipliers were chosen for comparison: *ALM* − -*SOA*10 [19], *ALM* − -*SOA*11 [19], *Mitchell multiplier* [14], *Mitchell* − -*trunc*6 [17], *Mitchell* − -*trunc*8 [17], *S* − -*ROBA* [24], *AS* − -*ROBA* [24], *DRUM*6 [20], and *ILM* − -*AA* [26].

The selected multipliers have been implemented in Verilog and synthesized using the TSMC 180-nm standard cell library. To evaluate the power, we used relaxed timing with 10 MHz virtual clocks with a 5% signal toggle rate and output load capacitance equal to 10 fF. The aim of these synthesis conditions is proof of logic concept and relative comparison of different approaches and to keep equal conditions for all experiments. Table 5 contains the synthesis results in terms of delay, power, area, and PDP, as well as NMED for all examined multipliers.

The results clearly show the impact of datapath pruning on the delay, area, and power. Datapath pruning reduces the size of the employed barrel shifters, which in turn shortens the delay and decreases the power consumption. We can see that *LOBO*12 − -12/8 achieves a similar delay as the fastest approximate non-logarithmic multipliers. The LOBO multipliers achieve a smaller delay than approximate logarithmic multipliers except for *ALM* − -*SOA*11, *ILM* − -*AA*, and *Mitchell* − -*trunc*6, which have significantly higher NMED. Although slightly faster, all R4ABM multipliers are characterized by the largest area utilization, and they consume twice as much energy as LOBO.

Fig. 6 reveals a correlation between area utilization and NMED for all examined multipliers. Compared with the approximate non-logarithmic multipliers, the LOBO designs provide substantially lower area utilization. In comparison with the approximate logarithmic multipliers, the multipliers presented in [17], [19] and [26] deliver the smallest area utilization at the expense of a significantly higher NMED.

Results obtained for PDP indicate the energy efficiency of LOBO designs. For example, the *LOBO*12 − -8/8 and *LOBO*12 − -12/8 multipliers offer smaller PDP than the approximate non-logarithmic multipliers, while keeping a small NMED. On the other hand, some approximate logarithmic multipliers deliver smaller PDP at the expense of a significantly higher NMED. Fig. V depicts the correlation between PDP and NMED for all examined multipliers. The LOBO designs with $d = 12$ occupy the space between

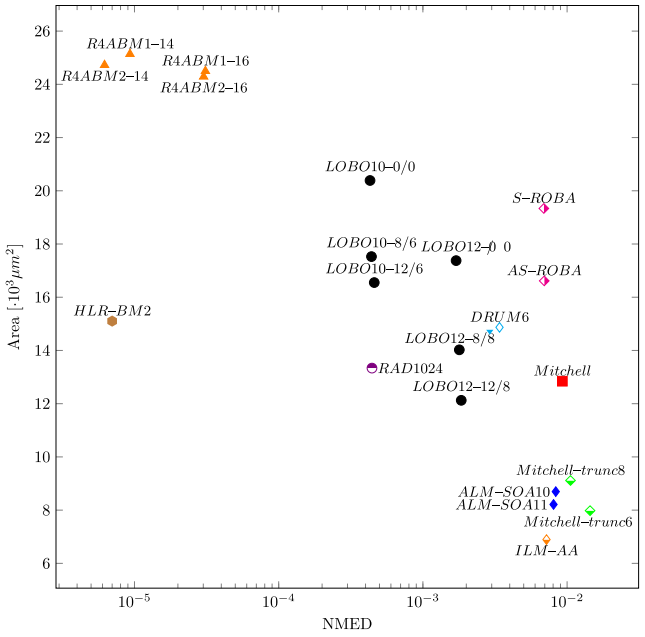


FIGURE 6. Area vs. NMED for the examined multipliers.

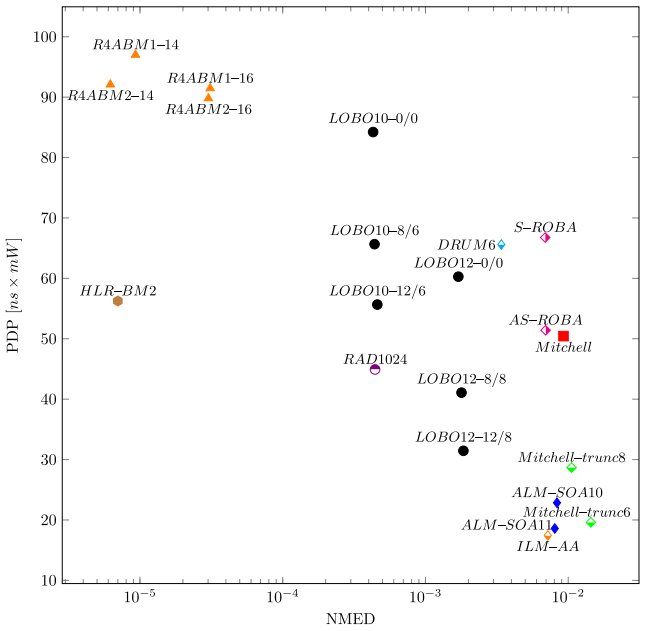


FIGURE 7. PDP vs. NMED for the examined multipliers.

the non-logarithmic and approximate logarithmic multipliers, denoting the potentially good compromise between NMED and PDP.

We also explore the reductions in area utilization and PDP of the examined multipliers compared to an exact radix-4 Booth multiplier. Fig. V shows area reduction vs. PDP reduction for all examined multipliers. We can see that the proposed approach (*LOBO*12 − -12/8) offers up to 75 % reduction in PDP and up to 50 % savings in area utilization. Only *Mitchell* − *-trunc* [17], *ALM* − *-SOA* [19] and

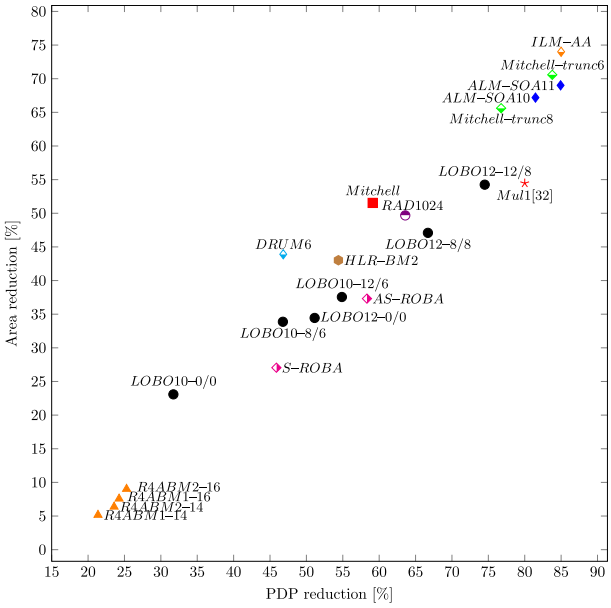


FIGURE 8. Area reduction vs. PDP reduction for the examined multipliers.

ILM − *-AA* [26] provide a greater reduction in area utilization and PDP than the proposed multipliers but they exhibit a quite large NMED.

Finally, we compare the proposed multiplier with the state-of-the-art multipliers based on inexact compressors [28], [32], [35], [36]. As these multipliers are 8-bit designs or implemented in FinFET technology, we compare them only in terms of PDP reductions and NMED reported in [28], [32], [35], [36]. Table 6 shows the reported NMED and PDP reduction for *LOBO*12−-12/8 the state-of-the-art multipliers based on inexact compressors. We can see that *LOBO*12 − -12/8 delivers smaller NMED at a similar PDP reduction as the state-of-the-art multipliers based on inexact compressors.

TABLE 6. Reported PDP reductions and NMED for *LOBO*12 − -12/8 and the state-of-the-art multipliers based on inexact compressors.

Multiplier	Width [*bits*]	PDP reduction [%]	NMED [$\cdot 10^{-4}$]
*LOBO*12–12/8	16	74.5	18.45
Reddy et al. [28]	8	68.0	98.00
Multiplier 1 [32]	16	80.0	178.00
Multiplier 2 [32]	16	39.8	0.071
Multiplier 1 [35]	8	60.4	190.00
Multiplier 2 [35]	8	65.5	180.00
Sabetzadeh et al. [36]	8	78.1	70.00

THE SCALABILITY OF THE LOBO MULTIPLIER

In this subsection, we discuss the scalability of the proposed multiplier when the multiplier's bit-width is 24 and 32 bit. We set $d = 20$ for 24-bit LOBO, and $d = 28$ for 32-bit LOBO. In such a way, we keep the same number of partial products as in LOBO12. For the 24-bit and 32-bit multiplier, LOBO20 and LOBO28 deliver the same MRE

as LOBO12 for 16-bit width. While LOBO12 delivers 50% fewer partial products, LOBO20 and LOBO28 deliver 66% and 75% fewer partial products than the exact multiplier, respectively. LOBO20's and LOBO28's partial product generation stage consists of only two Radix-4 Booth encoders and an LPPG stage. On the other hand, the 24-bit exact radix-4 multiplier has 12 Booth encoders, while the 32-bit exact radix-4 multiplier has 16 Booth encoders. For the radix-4 Booth encoding, the scaling factor of the product generation stage equals $\mathcal{O}(n^2/2)$, where n denotes the bit width (n bits of multiplicand and $n/2$ partial products). The scalability of the LPPG stage is determined by the scalability of the barrel shifter, which equals to $\mathcal{O}(n \log n)$ [46].

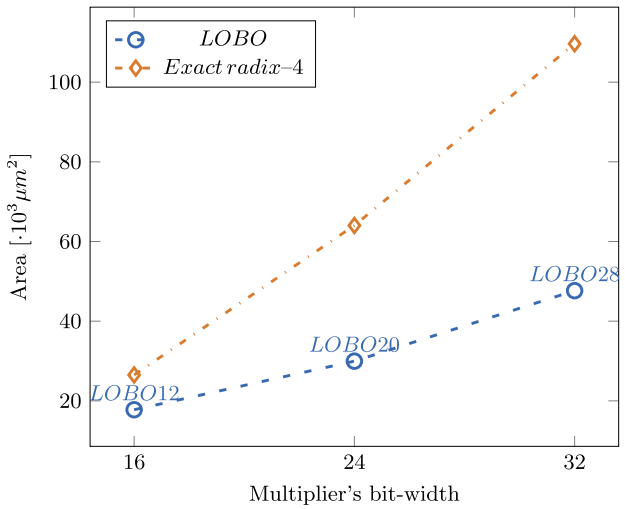


FIGURE 9. Area utilization for an exact radix-4 and the LOBO multipliers vs. multiplier bit-width.

Fig. 9 illustrates the growth in area utilization with the increase of multiplier's bit-width. It is evident from Fig. 9, that the LOBO multipliers have smaller growth in area utilization than exact radix-4 multipliers.

VI. APPLICATION CASE STUDIES

To assess the usability of the proposed LOBO multiplier, we examine the impact of approximate multipliers in image sharpening and Convolutional Neural Networks based image classification. Here, we examine the impact of *LOBO*12 − -12/8 and four state-of-the-art approximate multipliers: *RAD*1024 [42], *AS* − *-ROBA* [24], *Mitchell* − *-trunc*6 [17], *ILM* − *-AA* [26], and *ALM* − *-SOA*10 [19]. Note that the *LOBO*12 − -12/8 multiplier has the largest NMED and the smallest PDP among all LOBO multipliers. Even though we have not assessed the impact of other LOBO multipliers with a smaller NME, we expect them to perform the same or even better than *LOBO*12 − -12/8.

A. IMAGE SHARPENING

The goal of image sharpening is to highlight the fine details in an image or to enhance the details that have been blurred [47]. In image sharpening, each pixel is obtained as follows:

$$Y(i,j) = \frac{1}{256} \sum_{m=-2}^{2} \sum_{n=-2}^{2} X(i+m, j+n) M_S(i+3, j+3), \tag{19}$$

where $Y(i,j)$ and $X(i,j)$ denote the output and input pixels in the i-th column and j-th row, respectively, while M_S is a 5×5 sharpening mask given as:

$$M_S = \begin{bmatrix} -1 & -4 & -6 & -4 & -1 \\ -4 & -16 & -24 & -16 & -4 \\ -6 & -24 & 476 & -24 & -6 \\ -4 & -16 & -24 & -16 & -4 \\ -1 & -4 & -6 & -4 & -1 \end{bmatrix} \tag{20}$$

The pixels in an input image $X(i,j)$ are uniformly shifted from [0,65535] to [−32768,32767], and represented as 16-bit integers. To asses the influence of product approximation, we employ the mean structural similarity index (MSSIM) [48] and the peak-signal-to-noise ratio (PSNR) between an image filtered with the exact multiplier and an image filtered with an approximate multiplier. The tests are performed on five standard color images from USC-SIPI Image Database [49]: *Airplane*, *Female*, *House*, *Mandrill* and *Peppers*.

Table 7 reports MSSIM and PRNS for image sharpening. High values of MSSIM and PSNR indicate that the product approximation has a negligible influence on image processing. The proposed *LOBO*12 − -12/8 design delivers the best PSNR, while *RAD*1024 and *AS* − *-ROBA* deliver lower PSNR and almost the same MSSIM. *ALM* − *-SOA*11, *ILM* − *-AA*, and *Mitchell* − *-trunc*6 deliver the worst PSNR because they have the largest NMED among all compared designs.

B. IMAGE CLASSIFICATION WITH CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural networks (CNN) represent the method of choice for various image classification tasks [50]. In the experiments with CNNs, we used the Caffe framework [51], where we replaced the exact multiplication with an approximate one in the convolutional and fully connected layers. Particularly, we replaced the calls to the cuBLAS multiplication routines used in the convolution and fully connected layers in the Caffe framework with the calls to our C/C++ routines which implement various approximate multiplier designs. We used approximate multiplication only in the inference pass.

For experiments with CNNs, we trained the cuda-convnet[1] network on the CIFAR10 dataset [52] using floating-point arithmetic. In our previous research [22], we showed that for the training of neural networks with integer (approximate) multipliers, we need at least 18 bits to represent the

[1]see `https://code.google.com/p/cuda-convnet/` for details on the model

TABLE 7. The MSSIM and PSNR and values for the image sharpening application.

	Airplane		*Female*		*House*		*Mandrill*		*Peppers*	
	MSSIM	PSNR[dB]	MSSIM	PSNR[dB]	MSSIM	PSNR[dB]	MSSIM	PSNR[dB]	MSSIM	PSNR[dB]
$LOBO12\text{–}12/8$	0.98	42.54	0.98	42.65	0.99	41.92	1.00	40.01	1.00	40.26
$ALM\text{–}SOA10$ [19]	1.00	33.31	1.00	29.72	1.00	35.73	1.00	32.48	0.99	33.79
$Mitchell\text{–}trunc6$ [17]	1.00	31.64	1.00	28.72	1.00	31.45	1.00	31.77	0.99	34.30
$ILM\text{–}AA$ [26]	0.80	16.31	0.60	15.29	0.56	14.54	0.52	12.61	0.53	14.35
$AS\text{–}ROBA$ [24]	0.99	35.00	1.00	43.94	0.99	34.80	1.00	36.96	0.99	33.79
$RAD1024$ [42]	0.97	40.07	0.96	41.37	1.00	40.68	1.00	39.32	1.00	32.39

weights and input signals for efficient use of the classical gradient-descent algorithm. For cuda-convnet, we performed training for 75.000 iterations using stochastic gradient descent with the step-down decay of the learning rate. To speed up training and to keep initial activations in a reasonable range, the average over the training set should be close to zero [53]. Thus, during the training and testing of cuda-convnet, we performed mean subtraction (the mean pixel value obtained on the whole CIFAR10 dataset was subtracted across every individual image). We employed the train/test split as defined in [52]. The training was repeated ten times with random weight initialization. Then, for each of the ten obtained weight models, we performed inference with approximate multiplication on all images from the test set and reported means and standard deviation of classification accuracy. This way, we discarded the influence of weight initialization on the classification results. During each inference phase, we used the 16-bit signed fixed-point format to represent all inputs, weights, and neuron outputs. For a floating-point number x we perform fixed-point conversion as follows:

$$x_q = \frac{\lfloor x \cdot 2^Q \rfloor}{2^Q}, \tag{21}$$

where x_q is a 16-bit signed fixed-point number, and Q denotes the length of the fractional part in the fixed-point representation. We chose $Q = 8$ for the cuda-convnet. This choice of Q delivers the smallest accuracy degradation when approximate multiplication is present. In case of overflow, x is converted to the largest or smallest value, depending on its sign.

Fig. 10 shows the obtained classification accuracy in cuda-convnet@CIFAR10 in the form of error bars. The results indicate that we can employ the $LOBO12 - 12/8$ design in the cuda-convnet@CIFAR10 without any noticeable degradation in classification accuracy. Furthermore, $LOBO12 - 12/8$, *AS-ROBA*, and *RAD1024* deliver the same accuracy as the exact 16-bit fixed-point multiplier, but $LOBO12 - 12/8$ has the smallest energy among them. On the other hand, *ALM-SOA11*, $ILM - AA$, and *Mitchell-trunc6*, which have the largest NMED and the smallest area and PDP among all examined multipliers, cause a significant drop in classification accuracy. The decrease in accuracy is too high to be justified by savings in energy and area offered by *ALM-SOA11*, $ILM - AA$, and $Mitchell - trunc6$.

In image processing and CNN image classification, we showed that LOBO multipliers deliver similar results

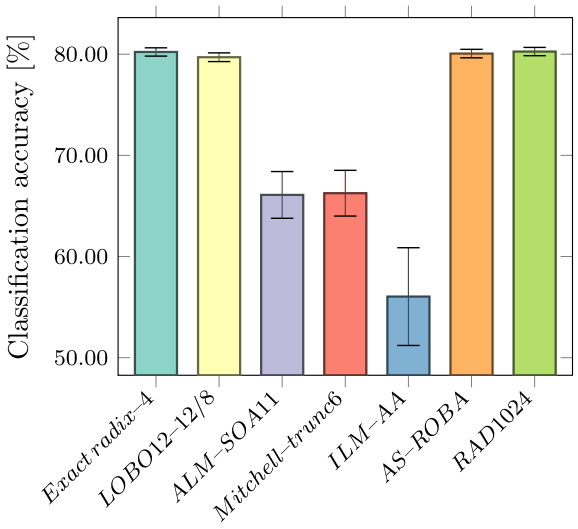


FIGURE 10. Classification accuracy in cuda-convnet@CIFAR10.

or outperform existing approximate multipliers. Instead of proposing a multiplier for one specific application, we aimed at making a highly customizable approximate multiplier, which covers a broader specter of applications that have different accuracy demands.

VII. CONCLUSIONS

In this paper, a novel approximate multiplier has been presented. The main idea of this multiplier is to encode the most-significant bits of the multiplier using exact radix-4 Booth encoding and to approximate the remaining high-radix partial product, which evolves from the least-significant bits, using an approximate logarithmic multiplier. A datapath pruning technique has been introduced to reduce the overall hardware complexity of the proposed multiplier. The presented multiplier is tunable in terms of accuracy and design efficiency using three parameters: d, T_H, and T_S.

We have carried out a set of experiments to assess the error, design efficiency, and applicability of the LOBO multiplier. The proposed design provides a good trade-off between approximation error and design efficiency. From the synthesis results, we can conclude that the datapath pruning has a low impact on multiplier accuracy, while it delivers significant savings in area utilization and power consumption. Thus, we conclude that datapath pruning is a powerful means of

achieving both accuracy and efficiency in logarithm-based multipliers.

Compared with the state-of-the-art approximate non-logarithmic multipliers, the proposed design offers lower energy consumption and area utilization, making it suitable in low-power applications. In contrast to the state-of-the-art approximate logarithmic multipliers, the proposed design provides better accuracy, which enables a broader area of usage. Besides, our studies showed that some state-of-the-art approximate logarithmic multipliers cause significant performance degradation, which cannot be justified with the gains in energy consumption. Previous work focused either on achieving good accuracy or energy-efficient design. Contrary to previous designs, the LOBO designs offer a compromise between low energy consumption and high accuracy. The LOBO's energy-accuracy compromise makes it the appropriate candidate for employment in the broader specter of error-resilient applications. The application studies support this claim, as the simplest approximate multipliers do not provide satisfactory accuracy in image processing and cuda-convnet. These findings reveal the potential of the proposed design strategy in the field of approximate multipliers.

In future research into the topic, we should investigate how different CNN layers tolerate a different amount of error. We believe that parametrized datapath pruning can be helpful to devise a strategy for placing different LOBO multipliers across a CNN architecture to suite the error tolerance of each layer.

ACKNOWLEDGMENT

The authors would like to thank Prof. Dušan Raič, Faculty of Electrical Engineering, University of Ljubljana, who performed the CMOS synthesis. They would also like to thank the retired professors Veselko Guštin, Dušan Kodek, and Ljubo Pipan, for encouraging them to persist in research in computer engineering and hardware and a colleagues, Prof. Uroš Lotrič and Prof. Iztok Lebar Bajec, who carefully read the manuscript and whose comments have further improved the final version of the article. Any remaining errors are, of course, our responsibility.

REFERENCES

[1] A. Agrawal, Z. Sura, J. Choi, K. Gopalakrishnan, S. Gupta, R. Nair, J. Oh, D. A. Prener, S. Shukla, and V. Srinivasan, "Approximate computing: Challenges and opportunities," in *Proc. IEEE Int. Conf. Rebooting Comput. (ICRC)*, Oct. 2016, pp. 1–8.

[2] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, p. 62, 2016.

[3] N. E. Jerger and J. S. Miguel, "Approximate computing," *IEEE Micro*, vol. 38, no. 4, pp. 8–10, Jul. 2018.

[4] L. Eeckhout, "Approximate computing, intelligent computing," *IEEE Micro*, vol. 38, no. 4, pp. 6–7, Jul. 2018.

[5] M. Alioto, V. De, and A. Marongiu, "Guest editorial energy-quality scalable circuits and systems for sensing and computing: From approximate to communication-inspired and learning-based," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 3, pp. 361–368, Sep. 2018.

[6] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proc. 18th IEEE Eur. TEST Symp. (ETS)*, May 2013, pp. 1–6.

[7] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. New York, NY, USA: Oxford Univ. Press, Inc., 2000.

[8] M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 10–14.

[9] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," 2014, *arXiv:1412.7024*. [Online]. Available: http://arxiv.org/abs/1412.7024

[10] S. Veeramachaneni, K. Krishna, L. Avinash, S. Puppala, and M. B. Srinivas, "Novel architectures for high-speed and low-power 3-2, 4-2 and 5-2 compressors," in *Proc. 20th Int. Conf. VLSI Design Held Jointly 6th Int. Conf. Embedded Syst. (VLSID)*, 2007, pp. 324–329.

[11] Z. Yang, J. Han, and F. Lombardi, "Approximate compressors for error-resilient multiplier design," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFTS)*, Oct. 2015, pp. 183–186.

[12] M. Kumm and J. Kappauf, "Advanced compressor tree synthesis for FPGAs," *IEEE Trans. Comput.*, vol. 67, no. 8, pp. 1078–1091, Aug. 2018.

[13] M. D. Ercegovac and T. Lang, *Digital Arithmetic*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann, 2003.

[14] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IEEE Trans. Electron. Comput.*, vols. EC-11, no. 4, pp. 512–517, Aug. 1962.

[15] V. Mahalingam and N. Ranganathan, "Improving accuracy in Mitchell's logarithmic multiplication using operand decomposition," *IEEE Trans. Comput.*, vol. 55, no. 12, pp. 1523–1535, Dec. 2006.

[16] S. Gandhi, M. S. Ansari, B. F. Cockburn, and J. Han, "Approximate leading one detector design for a hardware-efficient mitchell multiplier," in *Proc. IEEE Can. Conf. Electr. Comput. Eng. (CCECE)*, May 2019, pp. 1–4.

[17] M. S. Kim, A. A. D. Barrio, L. T. Oliveira, R. Hermida, and N. Bagherzadeh, "Efficient Mitchell's approximate log multipliers for convolutional neural networks," *IEEE Trans. Comput.*, vol. 68, no. 5, pp. 660–675, May 2019.

[18] H. Kim, M. S. Kim, A. A. Del Barrio, and N. Bagherzadeh, "A cost-efficient iterative truncated logarithmic multiplication for convolutional neural networks," in *Proc. IEEE 26th Symp. Comput. Arithmetic (ARITH)*, Jun. 2019, pp. 108–111.

[19] W. Liu, J. Xu, D. Wang, C. Wang, P. Montuschi, and F. Lombardi, "Design and evaluation of approximate logarithmic multipliers for low power error-tolerant applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 9, pp. 2856–2868, Sep. 2018.

[20] S. Hashemi, R. I. Bahar, and S. Reda, "DRUM: A dynamic range unbiased multiplier for approximate applications," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2015, pp. 418–425.

[21] Z. Babić, A. Avramović, and P. Bulić, "An iterative logarithmic multiplier," *Microprocessors Microsystems*, vol. 35, no. 1, pp. 23–33, Feb. 2011.

[22] U. Lotrič and P. Bulić, "Applicability of approximate multipliers in hardware neural networks," *Neurocomputing*, vol. 96, pp. 57–65, Nov. 2012.

[23] S. E. Ahmed and M. B. Srinivas, "An improved logarithmic multiplier for media processing," *J. Signal Process. Syst.*, vol. 91, no. 6, pp. 561–574, Jun. 2019.

[24] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram, "RoBA multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 2, pp. 393–401, Feb. 2017.

[25] S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram, "TOSAM: An energy-efficient truncation- and rounding-based scalable approximate multiplier," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 5, pp. 1161–1173, May 2019.

[26] M. S. Ansari, B. F. Cockburn, and J. Han, "A hardware-efficient logarithmic multiplier with improved accuracy," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 928–931.

[27] A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of approximate compressors for multiplication," *IEEE Trans. Comput.*, vol. 64, no. 4, pp. 984–994, Apr. 2015.

[28] K. M. Reddy, M. H. Vasantha, Y. B. N. Kumar, and D. Dwivedi, "Design and analysis of multiplier using approximate 4-2 compressor," *AEU-Int. J. Electron. Commun.*, vol. 107, pp. 89–97, Jul. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1434841118330085

[29] C.-H. Lin and I.-C. Lin, "High accuracy approximate multiplier with error correction," in *Proc. IEEE 31st Int. Conf. Comput. Design (ICCD)*, Oct. 2013, pp. 33–38.

[30] M. Ha and S. Lee, "Multipliers with approximate 4–2 compressors and error recovery modules," *IEEE Embedded Syst. Lett.*, vol. 10, no. 1, pp. 6–9, Mar. 2018.
[31] H. Jiang, C. Liu, F. Lombardi, and J. Han, "Low-power approximate unsigned multipliers with configurable error recovery," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 1, pp. 189–202, Jan. 2019.
[32] S. Venkatachalam and S.-B. Ko, "Design of power and area efficient approximate multipliers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 5, pp. 1782–1786, May 2017.
[33] I. Qiqieh, R. Shafik, G. Tarawneh, D. Sokolov, S. Das, and A. Yakovlev, "Significance-driven logic compression for energy-efficient multiplier design," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 3, pp. 417–430, Sep. 2018.
[34] D. Esposito, A. G. M. Strollo, E. Napoli, D. De Caro, and N. Petra, "Approximate multipliers based on new approximate compressors," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 12, pp. 4169–4182, Dec. 2018.
[35] M. Ahmadinejad, M. H. Moaiyeri, and F. Sabetzadeh, "Energy and area efficient imprecise compressors for approximate multiplication at nanoscale," *AEU-Int. J. Electron. Commun.*, vol. 110, Oct. 2019, Art. no. 152859. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1434841119304923
[36] F. Sabetzadeh, M. H. Moaiyeri, and M. Ahmadinejad, "A majority-based imprecise multiplier for ultra-efficient approximate image multiplication," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 11, pp. 4200–4208, Nov. 2019.
[37] H. Jiang, J. Han, F. Qiao, and F. Lombardi, "Approximate radix-8 booth multipliers for low-power and high-performance operation," *IEEE Trans. Comput.*, vol. 65, no. 8, pp. 2638–2644, Aug. 2016.
[38] H. Waris, C. Wang, and W. Liu, "Hybrid low radix encoding based approximate booth multipliers," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, early access, Feb. 19, 2020, doi: 10.1109/TCSII.2020.2975094.
[39] W. Liu, L. Qian, C. Wang, H. Jiang, J. Han, and F. Lombardi, "Design of approximate radix-4 booth multipliers for error-tolerant computing," *IEEE Trans. Comput.*, vol. 66, no. 8, pp. 1435–1441, Aug. 2017.
[40] M. S. Ansari, H. Jiang, B. F. Cockburn, and J. Han, "Low-power approximate multipliers using encoded partial products and approximate compressors," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 3, pp. 404–416, Sep. 2018.
[41] G. Zervakis, S. Xydis, K. Tsoumanis, D. Soudris, and K. Pekmestzi, "Hybrid approximate multiplier architectures for improved power-accuracy trade-offs," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, Jul. 2015, pp. 79–84.
[42] V. Leon, G. Zervakis, D. Soudris, and K. Pekmestzi, "Approximate hybrid high radix encoding for energy-efficient inexact multipliers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 3, pp. 421–430, Mar. 2018.
[43] V. Leon, G. Zervakis, S. Xydis, D. Soudris, and K. Pekmestzi, "Walking through the energy-error Pareto frontier of approximate multipliers," *IEEE Micro*, vol. 38, no. 4, pp. 40–49, Jul. 2018.
[44] W. Liu, T. Cao, P. Yin, Y. Zhu, C. Wang, E. E. Swartzlander, and F. Lombardi, "Design and analysis of approximate redundant binary multipliers," *IEEE Trans. Comput.*, vol. 68, no. 6, pp. 804–819, Jun. 2019.
[45] D. Jacobsohn, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, no. 6, p. 754, Dec. 1964.
[46] J. A. Fisher, P. Faraboschi, and C. Young, *Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools*. Amsterdam, The Netherlands: Elsevier, 2005.
[47] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed. Reading, MA, USA: Addison-Wesley, Mar. 1992, pp. 464–465.
[48] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004.
[49] A. Weber, "The USC-SIPI image database: Version 6," Dept. Elect. Eng., Signal Image Process. Inst., Univ. Southern California, Los Angeles, CA, USA, Tech. Rep. USC-SIPI-432, Feb. 2018.
[50] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[51] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," 2014, *arXiv:1408.5093*. [Online]. Available: http://arxiv.org/abs/1408.5093
[52] Y. A. LeCun, L. Bottou, G. B. Orr, and K. R. Müller, "Efficient BackProp," in *Neural Networks: Tricks of the Trade* (Lecture Notes in Computer Science), vol. 7700, G. Montavon, G. B. Orr, and K. R. Müller, Eds. Berlin, Germany: Springer, Dec. 1996.
[53] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade*. London, U.K.: Springer-Verlag, 1998, pp. 9–50. [Online]. Available: http://dl.acm.org/citation.cfm?id=645754.668382

RATKO PILIPOVIĆ (Member, IEEE) received the B.Sc. and M.Sc. degrees from the Faculty of Electrical Engineering, University in Banjaluka, Bosnia and Hercegovina, in 2015 and 2017, respectively. He is currently pursuing the Ph.D. degree with the Faculty of Computer and Information Science, University of Ljubljana, Slovenia. His research interests include approximate computing, arithmetic circuit design, FPGA design, embedded processing, and machine vision.

PATRICIO BULIĆ (Member, IEEE) received the B.Sc. degree in electrical engineering and M.Sc. and Ph.D. degrees in computer science from the University of Ljubljana, Slovenia, in 1998, 2001, and 2004, respectively. He is currently an Associate Professor with the Faculty of Computer and Information Science, University of Ljubljana. His main research interests include computer architecture, digital design, approximate computing, computer arithmetics, and parallel processing.