IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Received 13 February 2022; revised 4 June 2022; accepted 15 June 2022.
Date of publication 30 June 2022; date of current version 6 September 2022.

Digital Object Identifier 10.1109/TETC.2022.3186240

Approximate Recursive Multipliers Using Low Power Building Blocks

EFSTRATIOS ZACHARELOS, ITALO NUNZIATA, GERARDO SAGGESE, ANTONIO G.M. STROLLO, (Senior Member, IEEE), AND ETTORE NAPOLI, (Senior Member, IEEE)

Efstratios Zacharelos, Italo Nunziata, Gerardo Saggese, and Antonio G.M. Strollo are with the Department of Electrical Engineering and Information Technology, University of Napoli Federico II, 80125 Naples, Italy
Ettore Napoli is with the Department of Information and Electrical Engineering and Applied Mathematics, University of Salerno, 84084 Fisciano, Italy
CORRESPONDING AUTHOR: EFSTRATIOS ZACHARELOS (efstratios.zacharelos@unina.it)

ABSTRACT Approximate computing, frequently used in error tolerant applications, aims to achieve higher circuit performances by allowing the possibility of inaccurate results, rather than guaranteeing a correct outcome. Many contributions target the binary multiplier aiming to minimize the complexity of this common yet power-hungry circuit. Approximate recursive multipliers are low-power designs that exploit approximate building blocks to scale up to their final size. In this paper, we present two novel 4×4 approximate multipliers obtained by carry manipulation. They are used to compose 8×8 designs with different error-performance trade-off. The final circuits exhibit a competitive behavior in terms of error while reducing the power dissipation when compared to state-of-the-art proposals. The proposed multipliers and state-of-the-art designs found in the literature, have been synthesized targeting a 14nm FinFET technology to determine the electrical characteristics. Compared with an exact 8×8 multiplier, the least dissipative design proposed in this paper reduces power consumption and silicon area by 46%, and minimum delay by 21%. It also consumes 14% less power than the least power-hungry recursive circuit found in the literature, while offering 81% higher accuracy. Image processing applications and a convolutional neural network are shown to demonstrate the effectiveness of the proposed multipliers.

INDEX TERMS Approximate methods, arithmetic and logic structures, error handling and recovery, integrated circuits

I. INTRODUCTION

The everlasting demand for power and speed improvement has driven researchers towards approximate computing. Approximate computing is a fast-emerging field in digital design that sacrifices the exactness of computations over significant improvement in power dissipation, speed, and circuit area. Such techniques can be utilized in cloud computing, embedded and mobile devices, where high speed and power minimization are important constraints. Approximate computing finds fertile soil in error resilient applications such as multimedia processing, data mining and recognition, machine learning [1]–[4].

Concerning approximate computing, a plethora of studies has focused on arithmetic operations, such as binary addition, multiplication, and division [5]–[7]. Binary multipliers constitute a fundamental part of digital processing systems, and unfortunately are characterized by heavy silicon area, power, and timing requirements [8]. Consequently, nowadays approximate binary multipliers are being studied thoroughly. A comprehensive survey of arithmetic circuits, such as approximate adders, multipliers, and more complex circuits such as the binary divider is reported in [9].

Several techniques providing efficient approximate multipliers have been studied in the literature. One such example is the approximate logarithmic multiplier [10]–[12]. In this case, approximated versions of the logarithms of the input operands, are added. The result corresponds to the approximated value of the antilogarithm of the sum. These are low power and high speed designs, due to the low complexity in their architecture. However, they tend to be less accurate. Another approach is the static segmentation. In this technique, a part of each input operand is given as input to a small multiplier, whose shifted output is the result of the multiplication [13]. Static segmentation has been demonstrated

to be useful when very low power is needed, and accuracy is not the main issue. In [14] the authors propose an approximate multiplier that can dynamically control accuracy. The circuit can select the length of the carry propagation to effectively satisfy the desired accuracy requirements.

Software-based approaches have been proposed, that merge the approximated multiplier design in the design flow of the circuit. They automatically generate synthesizable hardware description code for approximate arithmetic circuits based on the accuracy requirement of the design [15]–[18]. Such techniques can prove useful when the targeted application does not have a uniform input distribution.

The basic binary multiplication process can be divided into three parts: partial product generation, partial product reduction and carry-propagate addition. Approximate computing can be introduced in all these steps. For instance, the first step can be approximated by truncating some of the least significant partial products (PPs) and then employing a compensation strategy [19], [20].

The partial product reduction step is typically the main target for approximations in a binary multiplier. A common approach to reduce the partial product matrix (PPM) relies on the use of approximate compressors. Compressors are logic circuits that aim to minimize the number of operands in the final step, which is the addition of the reduced partial products. They are XOR-rich circuits (thus slow and power-hungry), that count the number of ones in the input. The most basic exact compressor is the Full Adder, that reduces three digits into two, maintaining the original information. Many research contributions have focused on the approximation of the PPM compression phase, [21]–[35].

In [21] the authors acquire approximate compressors by truncating outputs of some exact compressors, while in [22] and [25], compressors with only 2-bit outputs are proposed. Lossy compression of the rows in the PPM based on bit significance, is investigated in [23]; the compression exploits approximate, OR-based half adders. In [24] simple OR gates serve as approximate compressors and two designs are proposed. The two designs are obtained using encoded partial products and approximate compressors, delivering different accuracy-electrical performance trade-off. Several solutions employing 3:2 and 4:2 compressors to generate approximated multipliers are presented in [26], [29], [31], [32]. A set of Single-Weight Approximate Compressors (SWACs) is employed in [27], to construct approximate multipliers. Unlike the Full-Adder that produces a sum and a carry, these designs compress input bits derived from a PPM column, into fewer output bits, maintaining the same initial weight. This allows a significant reduction of circuit complexity since less carry bits are generated and propagated. Maddisetty *et al.* [28] present the training of a neural network to devise an efficient approximate 4:2 compressor. In [30] two 4:2 compressors are presented; a novel 4:2 architecture, and a modified design by substituting the AND / OR gates with NAND / NOR gates respectively. Although the boolean expression is changed, when the modified version targets multipliers, employing reduction steps in multiples of 2, the difference is nullified. Approximate 4:2 designs implemented in FinFET technology are presented in [33], [35]. In [34] the number of outputs of the approximate 4:2 compressor is innovatively reduced to one; 3 such compressors are proposed, as well as an error-correcting module.

Recursive multipliers are an interesting research area of the approximate computing field that aims to use small elementary approximate multiplier blocks, suitably assembled, to design larger multipliers, [26], [36]–[42]. The advantage of the recursive building of larger multipliers is that it avoids a dedicated design for every bit-width and gains in terms of generality of the proposed approaches. As explained in [26], four $n\times n$ building blocks can be utilized to scale up to a $2n\times 2n$ multiplier. Several authors have used 4×4 approximate multipliers to recursively generate several 8×8 multiplier alternatives. The authors of [26] propose three 4:2 compressors, used to generate two 4×4 multipliers.

Guo *et al.* [38] propose a 4×4 approximate multiplier module. The corresponding 8×8 multiplier is made up from one 4×4 multiplier featuring OR-based compressors with no carry propagation in the lower part, two of the proposed 4×4 modules in the middle part, and an exact 4×4 multiplier for the most significant part. Differently from the other designs, the four products are summed using an approximate adder.

In [40] the authors consider the probability distribution of the input operands to propose 4×4 multipliers, consisting of approximate NOR-based half adder and full adder designs. These elementary blocks are exploited to build approximate recursive multipliers. In [41] a 4×4 approximate multiplier featuring an error detection and correction system, is presented.

Similarly, in [36], [37], [39] and [42] the authors propose 2×2 approximate sub-multipliers, suitably arranged, to form larger size multipliers. Sixteen 2×2 modules are needed to create an 8×8 multiplier. Kulkarni *et al.* [36] present a 2×2 inexact multiplier with tunable error characteristics. In [37] the authors provide an exploration of the architectural space and propose their 2×2 module. The 2×2 approximate multiplier presented in [39] has an internal self-healing strategy that does not require coupled modules, while the proposed larger multipliers derived from the 2×2 blocks produce near zero mean error. In [42] two elementary multipliers are proposed that exhibit double-sided error distribution while the resulting 8×8 design has the advantage of error compensation.

In this paper, two novel 4×4 approximate designs with minimal power requirements and competitive error performance, are presented. The output is calculated by exploiting carry truncation and compensation techniques. These designs, along with an OR-based and an exact 4×4 multiplier, are used to generate 8×8, 16×16, and 32×32, approximate multipliers, following the strategies presented in [26] and [38].

The circuits proposed in this paper as well as various previously proposed contributions, have been synthesized using a commercial 14nm FinFET standard cell library. Syntheses show that our circuits, compared to previously proposed designs, provide good error-electrical performance trade-off.

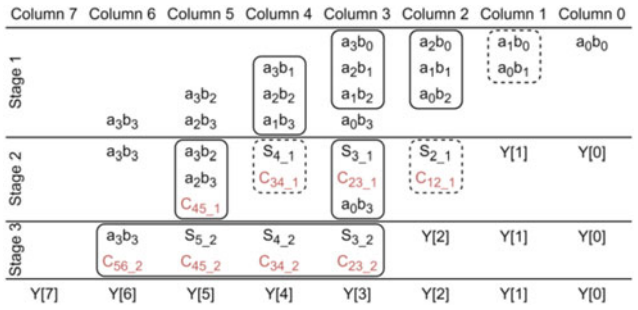

FIGURE 1. Wallace 4×4 exact multiplier.

We have also investigated the performance of 8×8 approximate multipliers in image filtering applications and in the inference step of a pre-trained convolutional neural network. Obtained results confirm that the proposed circuits are good competitors in error-resilient applications.

The paper is organized as follows. In Section II the approximate OR-based and proposed 4×4 multipliers are presented. The architectures of the recursive 8×8 multipliers are in Section III. Section IV reports the performances of 4×4, 8×8, 16×16, and 32×32 multipliers. Section V shows a comparison with formerly proposed approximate multipliers for image processing applications and for the inference stage of a pre-trained convolutional neural network. The conclusions are drawn in Section VI.

II. 4×4 APPROXIMATE BINARY MULTIPLIERS

Let us consider two 4-bit unsigned numbers $a = \sum_{i=0}^{3} a_i 2^i$ and $b = \sum_{j=0}^{3} b_j 2^j$. The computation of their product, $y = \sum_{k=0}^{7} y_k 2^k$, consists of three steps. Firstly, the partial product matrix (PPM) is generated using AND gates between all the input bits. There are various techniques to carry out the second and third steps that reduce and sum the entire PPM to obtain the final product, e.g., employing full adders, half adders or 4:2 compressors in Wallace or Dadda configurations. Figure 1 shows the Wallace reduction tree for an exact 4×4 multiplier. Three half adders (dashed rectangles) and five full adders (rectangles) are employed to reduce the PPM. The sum and carry outputs produced by half and full adders are indicated in the figure as *SN_x*, *CNM_x*, where *N* and *M* indicate the origin and destination column, while *x* indicates the reduction stage. After two stages of reduction we obtain the three least-significant bits of the output Y[2]...Y[0] and two 4 bit values that are summed to obtain the most significant bits of the output, Y[7]...Y[3].

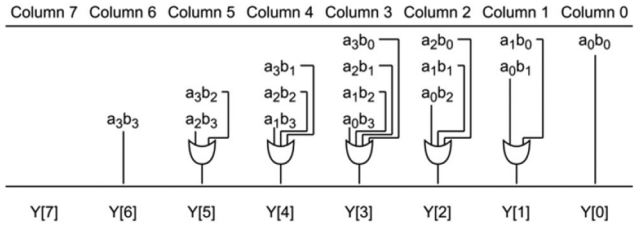

FIGURE 2. OR-based 4×4 approximate multiplier.

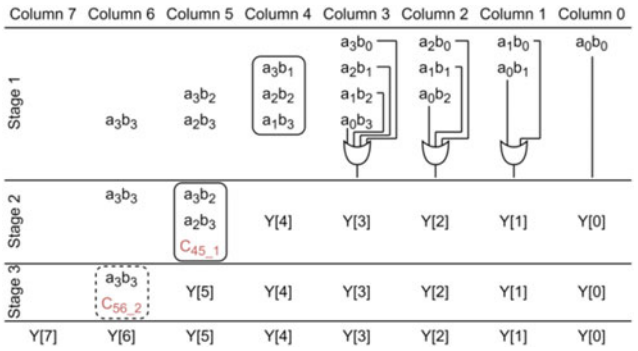

FIGURE 3. Half OR-based 4×4 approximate multiplier. The starting architecture for the proposed N1 design.

A simple and fast way to approximate the product of two binary numbers is to use an approximate multiplier with OR compressors. In this case all the partial products in each column of the PPM are fed to OR gates as shown in Figure 2. As it can be observed the most significant bit is always zero. This approximated design is a kind of lower bound for circuit complexity but, as shown in Table 3, it exhibits the worst error performance.

A. 4×4 APPROXIMATE MULTIPLIER N1

In the circuit shown in Figure 2, the sums of the partial products are approximated using OR gates. As a more accurate base point, we can assume an approximate multiplier that uses OR gates to sum the lower half of the matrix of partial products, and full or half adders for the higher part, as shown in Figure 3. Note that the approximated multiplier in Figure 3 requires three compression stages, while the OR based in Figure 2 obtains the result with a fast single stage.

The design in Figure 3 contains three XOR gates that are known to be bulky and slow. An attempt has been made to simplify it. The first step is to substitute the XOR gate in column 4 with a simpler OR gate:

$$Y[4] = a_3b_1 + a_2b_2 + a_1b_3 \tag{1}$$

The next step is the manipulation of the carry of the same full adder:

$$C_{45\_1} = a_3b_1 \cdot a_2b_2 + a_1b_3 \cdot a_2b_2 + a_3b_1 \cdot a_1b_3 \tag{2}$$

Let us simplify the expression by neglecting the last term:

$$C^*_{45\_1} = a_2b_2 \cdot (a_1b_3 + a_3b_1) \tag{3}$$

A customized Full Adder is employed in column 5 to add the three terms. The sum is exact and uses a XOR gate:

$$Y[5] = a_3b_2 \oplus a_2b_3 \oplus C^*_{45\_1} \tag{4}$$

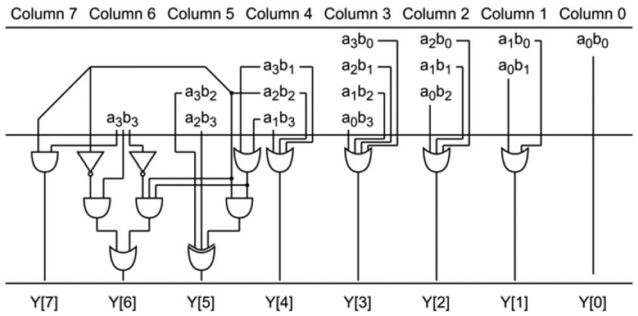

FIGURE 4. Proposed 4×4 approximate multiplier, N1.

The carry can be significantly simplified:

$$C_{56\_2} = a_3b_2 \cdot a_2b_3 + a_3b_2 \cdot C^*_{45\_1} + a_2b_3 \cdot C^*_{45\_1} = a_2b_2 \cdot (a_3b_3 + a_3b_3a_1 + a_3b_3b_1 + a_3b_1 + a_1b_3) \quad (5)$$

By neglecting the terms that are a product of three literals (they have a lower probability of being '1') we get:

$$C^*_{56\_2} = a_2b_2 \cdot (a_3b_3 + a_3b_1 + a_1b_3) \quad (6)$$

The two terms in column 6 are fed into a customized half adder. The sum is the XOR of the two inputs:

$$Y[6] = a_3b_3 \oplus C^*_{56\_2} = a_3b_3 \cdot \overline{C^*_{56\_2}} + \overline{a_3b_3} \cdot C^*_{56\_2} \Rightarrow$$
$$Y[6] \cong a_3b_3 \cdot \overline{a_2b_2} + \overline{a_3b_3} \cdot a_2b_2 \cdot (a_3b_1 + a_1b_3) \quad (7)$$

Finally, the carry of the Half Adder is approximated as:

$$Y[7] = C^*_{56\_2} \cdot a_3b_3 \cong a_2b_2 \cdot a_3b_3 \quad (8)$$

The resulting design is named N1 and is shown in Figure 4. N1 uses three stages to reach the result and uses six OR gates, four AND gates, and one XOR gate. Compared to the exact Wallace 4×4 multiplier, it shows a vast improvement in terms of both power and speed. In fact, in the exact design the third stage consists of cascaded half and full adders, resulting in three sub-stages, all of them containing at least one XOR gate. Namely, 28 AND gates, 8 OR gates and 12 XOR gates are used in the exact design. Obviously, the proposed design

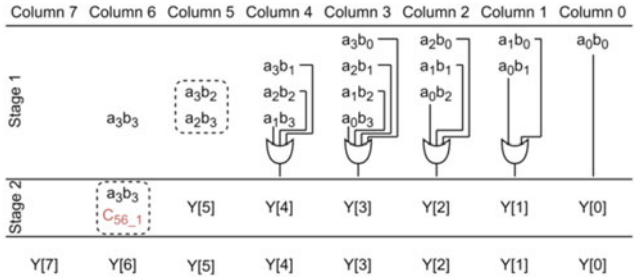

FIGURE 5. 4×4 approximate multiplier. The five least significant outputs are approximated with OR gates while the most significant three outputs are computed with two HAs. The starting architecture for the proposed N2 design.

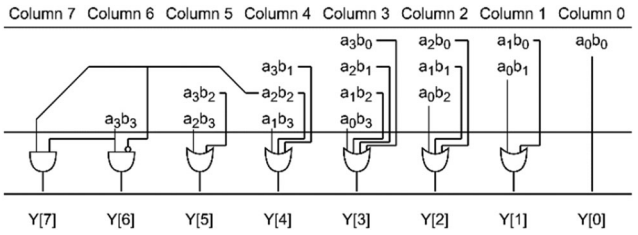

FIGURE 6. Proposed 4×4 approximate multiplier, N2.

provides an inexact result. The error characteristics of the proposed blocks are discussed in Section IV.

B. 4×4 APPROXIMATE MULTIPLIER N2

Let us now start from a less accurate circuit than the one in Figure 3. In this circuit, shown in Figure 5, all the terms from $Y[0]$ to $Y[4]$ are computed as the output of OR gates, while the remaining bits are computed without approximations. As shown in Figure 5, two half adders are needed together with the OR gates to complete the design of the multiplier. The proposed architecture takes the circuit in Figure 5 as a starting point for further simplification.

The first step is to substitute the XOR gate of the half adder in column 5, with an OR gate:

$$Y[5] = a_3b_2 + a_2b_3 \quad (9)$$

The carry of the same half adder is:

$$C_{56\_1} = a_3b_2 \cdot a_2b_3 \quad (10)$$

The sum of the last half adder is:

$$S_6 = a_3b_3 \oplus C_{56\_1} = a_3b_3 \cdot \overline{C_{56\_1}} + \overline{a_3b_3} \cdot C_{56\_1} = a_3b_3 \cdot \overline{a_3b_2 \cdot a_2b_3} + \overline{a_3b_3} \cdot a_3b_2 \cdot a_2b_3 \quad (11)$$

By neglecting the second term:

$$S^*_6 = a_3b_3 \cdot \overline{a_3b_3a_2b_2} = a_3b_3 \cdot \left(\overline{a_3b_3} + \overline{a_2b_2}\right) \quad (12)$$

With a final approximation:

$$Y[6] = a_3b_3 \cdot \overline{a_2b_2} \quad (13)$$

The carry of the last Half Adder is:

$$Y[7] = C_{56\_1} \cdot a_3b_3 = a_3b_2 \cdot a_2b_3 \cdot a_3b_3 = a_3b_3 \cdot a_2b_2 \quad (14)$$

The resulting design is named N2 and shown in Figure 6. This rather simple design has only two additional AND gates with respect to the OR-based design shown in Figure 2. However, the performances of the proposed design are considerably better as will be discussed in Section IV and shown in Table 3, making this design useful for higher order multipliers.

TABLE 1. 8×8 Approximate multiplier compositions exact sub-products addition.

Design		a_Hb_H	a_Lb_H	a_Hb_L	a_Lb_L
Proposed	N8-5	Exact	Exact	Exact	N1
	N8-6	Exact	Exact	Exact	N2
[26]	M8-1	M1	M1	M1	M1
	M8-2	M2	M2	M2	M2
	M8-3	Exact	M1	M1	M1
	M8-4	Exact	M2	M2	M2
	M8-5	Exact	Exact	Exact	M1
	M8-6	Exact	Exact	Exact	M2
[36]	Kul8	Kul4	Kul4	Kul4	Kul4
[37]	Reh8	Reh4	Reh4	Reh4	Reh4
[40]	Ax8_1	Exact	Exact	Exact	MxA
	Ax8_2	Exact	Exact	LxA	MxA
	Ax8_3	Exact	LxA	LxA	MxA
[42]	AxRM1	Exact	Exact	Exact	mul2b4
	AxRM2	Exact	Exact	mul2b4	mul2b4
	AxRM3	Exact	mul2a4	mul2b4	mul2b4

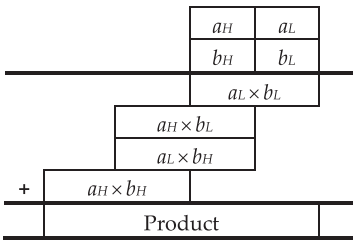

FIGURE 7. Recursive Multiplier using four building blocks.

III. 8×8 APPROXIMATE MULTIPLIER ARCHITECTURES

We compare our work to state-of-the-art designs in the literature [13], [14], [20], [26], [36]–[40], and [42]. These are mostly approximate recursive proposals, but contributions from other fields are also considered. In [13], [14], [20], and [39] no explicit 4×4 designs are proposed.

As mentioned in the introduction, scaling up to a $2n \times 2n$ multiplier can be achieved by exploiting four $n \times n$ multipliers. The same technique can be used recursively to design even larger multipliers. For instance, four suitably placed 2×2 multipliers form a 4×4 multiplier, while sixteen 2×2 multipliers can be used to generate an 8×8 design. Note that the building blocks do not need to be the same and different ones can be used, to obtain different electrical performance-accuracy trade-offs. As a rule of thumb, if uniform distribution is expected for the input operands, exact or high precision modules should occupy the most significant portion of the design. Moving towards the least significant part, modules that are less accurate, but also less demanding in terms of resources, might be used.

Consider two 8-bit unsigned numbers $a = \sum_{i=0}^{7} a_i 2^i$ and $b = \sum_{j=0}^{7} b_j 2^j$. In order to exploit recursive 4×4 multipliers to calculate the product $y = \sum_{k=0}^{15} y_k 2^k$, each number is divided into two 4-bit parts: $a_L = \sum_{i=0}^{3} a_i 2^i$, $a_H = \sum_{i=4}^{7} a_i 2^i$, $b_L = \sum_{i=0}^{3} b_i 2^i$ and $b_H = \sum_{i=4}^{7} b_i 2^i$ and the multiplications a_Lb_L, a_Hb_L, a_Lb_H, and a_Hb_H are performed exploiting the corresponding blocks. Finally the four sub-products need to be added. As shown in Figure 7, the four sub-products are added employing an exact adder.

TABLE 2. 8×8 Approximate multiplier compositions approximate sub-products addition.

Design		a_Hb_H	a_Lb_H	a_Hb_L	a_Lb_L
Proposed	N8-L1	Exact	N1	N1	OR-based
	N8-L2	Exact	N2	N2	OR-based
[38]	LOAM	Exact	guo4	guo4	OR-based

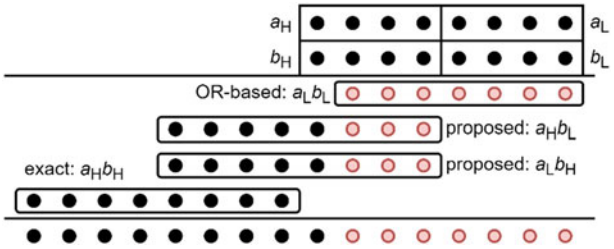

FIGURE 8. Proposed 8×8 approximate multiplier architecture. Red bits are added with OR gates, black bits with exact adders.

Table 1 shows the circuits considered for comparison that apply this design methodology ([26], [36], [37], [40] and [42]), the corresponding 4×4 building blocks, and how they are used to build larger multipliers. The multiplier names reported in the Table are directly taken from the reference papers. Note that the 4×4 approximate modules used in [42], namely mul2a4 and mul2b4, are also recursive multipliers made up by 2×2 blocks.

Table 1 also shows the composition of two of the four 8×8 multipliers proposed in this paper, namely N8-5 and N8-6. They use the proposed N1 and N2 blocks solely in the least significant part of the multiplier and produce fairly accurate results. As it will be shown in the following, they overcome the state-of-the-art when compared with other proposals in the same error range.

The circuit proposed in [39], that is used as a comparison in this paper, is not shown in Table 1, since it exploits 2×2 approximate multipliers to scale directly up to 8×8, without proposing specific 4×4 building blocks.

An alternative way to add the sub-products is proposed in [38] and used also in this paper. The utilized building blocks and their positions are shown in Table 2. Differently from Figure 7, the final product is not the exact addition of the four sub-products, but an approximated version of it. As it can be seen in Figure 8, the seven least significant columns of the sub-products are marked with red color, indicating that they are summed using an approximate adder that uses one OR gate in every column. However, the nine most significant columns are added with an exact adder. Note that the first sub-product has only seven output bits as shown in Figure 2.

IV. PERFORMANCES

The proposed and reference circuits are all synthesized in a 14nm FinFET technology, using Cadence Genus and imposing proper timing constraints. Power dissipation is computed by simulating the final netlist with random inputs, to obtain the switching activity of each node. The input vector array is identical for all designs with the same input bit width. In the following tables "Min delay" refers to the strictest timing constraint, at which each circuit can be synthesized with non-negative slack and provides information regarding the maximum working speed of each design.

Area, power, and delay are compared against the results of the corresponding (4×4, 8×8, 16×16, or 32×32) exact multiplier. The exact design is obtained by describing the circuit in HDL with the multiplication operator and letting the synthesizer choose the near-optimal topology for the given constraint. Therefore, the electrical performances are sometimes slightly worse than those presented in the literature that compare with a fixed exact design.

Error performance is obtained by an exhaustive simulation, for both 4×4 and 8×8 multiplier designs. For 16×16 and 32×32 designs the error performances are computed using a random set of uniformly distributed test vectors. The numbers of test vectors are 10^5 and 10^6 for the 16 and 32 bit multipliers, respectively.

The error metrics that are used in this paper are listed in the following. Let Y_{E_i} be the exact result of the multiplication between the two n-bits operands A_i and B_i such that $Y_{E\_i} = A_iB_i$ and let Y_{A_i} be the approximated output returned by the investigated inexact multiplier. The error E_i, of each multiplication is given by:

$$E_i = Y_{E\_i} - Y_{A\_i} \tag{15}$$

While the error distance ED_i is defined as:

$$ED_i = |Y_{E\_i} - Y_{A\_i}| \tag{16}$$

And the relative error distance RED_i, as:

$$RED_i = ED_i / Y_{i,EXACT} \forall Y_{i,EXACT} \neq 0 \tag{17}$$

1. The Normalized Mean Error Distance, NMED, is defined as the average value of ED divided by the maximum possible value returned by the multiplier, which is: $(2^{n-1})^2$.
2. The Mean Relative Error Distance, MRED, is given by the average value of *RED*.
3. The number of effective bits, *NoEB*, is defined as:

$$NoEB = 2n - \log_2(1 + \sqrt{E_{ms}}) \tag{18}$$

   where E_{ms} is the means square error, given by the average value of E^2.
4. The error rate, ER, is defined as the number of erroneous multiplications (with $E_i \neq 0$) over the total amount of possible inputs 2^{2n}.

A. 4×4 APPROXIMATE MULTIPLIERS

The electrical and error performances of the considered 4×4 approximate multipliers are summarized in Table 3. To ensure a fair comparison between the circuits, avoid biased optimizations by the synthesizing tool, and emphasize the low power performance of the structures, the circuits have been synthesized with the timing constraint of 250ps to obtain the area and power values. The circuits are simulated applying a uniformly distributed random set of $2{\cdot}10^4$ test vectors to gather the switching activity. The total power

TABLE 3. Performances of 4×4 approximate multipliers.

Design		Area* [μm²]	Power @1GHz Total [μW]	Power @1GHz Reduction [%]	Min Delay* [ps]	Error Rate [%]	NMED (×10⁻²)	MRED (×10⁻²)	NoEB
	Exact	17.27	25.07	-	115	-	-	-	8
	OR-Based	3.90	7.63	69.56	19	37.11	3.60	8.78	3.74
Proposed	N1	5.42	9.05	63.91	42	35.94	1.76	5.57	4.83
	N2	4.54	7.91	68.44	19	37.71	2.44	7.24	4.42
[26]	M1	8.85	13.93	44.44	51	35.94	1.75	6.08	4.88
	M2	6.01	9.55	61.90	24	35.94	2.76	7.87	4.20
[36]	Kul4	11.87	19.74	21.27	92	19.14	1.39	2.97	4.61
[37]	Reh4	16.69	22.05	12.06	105	46.48	2.08	15.90	4.71
[38]	guo4	9.57	17.79	29.03	72	28.52	1.89	4.57	4.46
[40]	MxA	6.36	9.50	62.09	30	53.91	6.99	22.80	3.16
	LxA	6.61	10.12	59.64	35	53.91	9.81	27.20	2.65
[42]	mul2a4	9.77	19.35	22.81	95	64.45	3.72	29.98	4.15
	mul2b4	10.79	18.27	27.13	81	75.00	7.46	51.38	3.36

**Area and power are reported for the circuits synthesized with a timing constraint of 250ps. min delay is the minimum timing at which the circuit can be synthesized with a non-negative slack.*

reported in the table is computed for a clock frequency of 1GHz. It is worth noting that the circuits proposed in this paper are for general purpose applications thus a uniform distribution of the input is considered. However, automated designs [15]–[18] or dedicated circuits previously presented in the literature, could provide better performances for a specific distribution of the input vectors.

As it can be observed in Table 3, the proposed circuits are very small and come second only to the OR-based design. The same can be stated also for power dissipation, with N2 having an unquestionable advantage. When it comes to speed, N2 is the fastest design while N1 is among the fastest. The proposed multipliers exhibit competitive NMED, MRED and NoEB with respect to the state-of-the-art. The relative reduction in power dissipation with respect to the exact design vs NoEB is shown in Figure 9. The proposed design N2 dissipates 18% less power than the least energy-hungry architectures up to date, M2 and MxA proposed in [26] and [40] respectively, while still providing a smaller approximation error.

B. 8×8 APPROXIMATE MULTIPLIERS

The results of the 8×8 approximate multipliers are shown in Table 4. Recursive designs are reported at the top part of the table, while selected approximate designs following different methodologies, are shown at the bottom part. Power reduction against number of effective bits for all designs is displayed in Figure 10. Non-filled shapes in the figure correspond to non-recursive designs.

All circuits are synthesized for a 1000ps timing constraint and simulated with the same set of $2{\cdot}10^4$ uniformly distributed random vectors. The total power reported in the table is computed for a clock frequency equal to 1GHz.

The designs presented in [13] employ a smaller, segmented multiplier. Specifically, instead of an 8-bit multiplier, a 4-bit multiplier with or without error correction respectively, is used. The product is then shifted accordingly. In this simple circuit, hardware resources and power consumption are kept to significantly low levels, while the error metrics are still competitive.

Note that the entries of [14] and [20] exhibit identical electrical performances respectively, since they refer to the same circuits with different settings (both designs allow for configurable accuracy). While the range of chosen accuracy in [14] is limited and the innate flexibility results in increased area requirements, the circuit is very fast, overcoming all the investigated contributions, including the proposed designs. As it can be observed in Table 4, the minimum accuracy of this design, is still greater than that of the design M8-2 proposed in [26], while power reductions are similar.

The circuit presented in [20] offers dynamic truncation at runtime, by enabling or disabling AND gates that form specific partial products. "DT0" refers to the case where all the AND gates are enabled, resulting in an exact multiplier. However, the additional hardware resources result in a greater power consumption with respect to the exact design (hence the negative power reduction). "DT8" refers to the maximum possible truncation where a 43.62% power reduction is achieved. The numbers in the names indicate the level of truncation.

The authors in [26], offer a number of circuits covering a wide range of accuracy. Designs M8-5 and M8-6 are the most precise ones, using one approximate and three exact 4-bit multipliers. While the synthesized circuits are slightly slower than the exact multiplier, they offer some power reduction at a relatively small expense in accuracy.

Designs Ax8_1 and AxRM1 presented in [40] and [42] respectively, employ three exact and one approximate modules. While these are the most accurate designs presented in the respective papers, they are still less accurate than M8-6 and M8-5 of [26], and even less accurate than the proposed N8-5 and N8-6. At the same time, the circuits are quite large, and slower than the exact multiplier. This behavior follows the pattern presented in Figure 9, for the 4×4 building

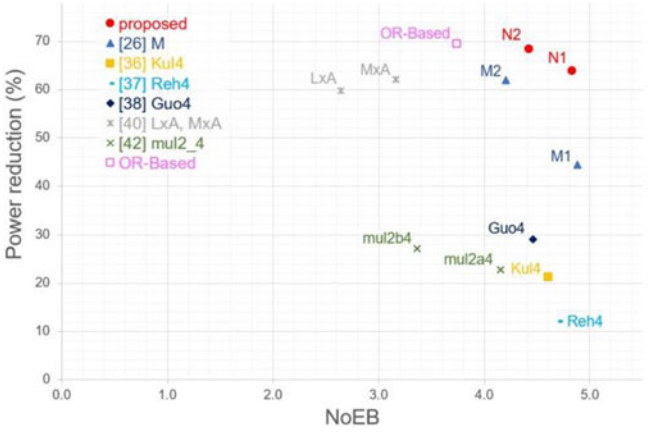

FIGURE 9. Power reduction of the considered 4×4 Approximate Multipliers with respect to the exact one vs Number of Effective Bits. The proposed circuits have lower power for the same NoEB. The exact design would have NoEB = 8 and zero power reduction.

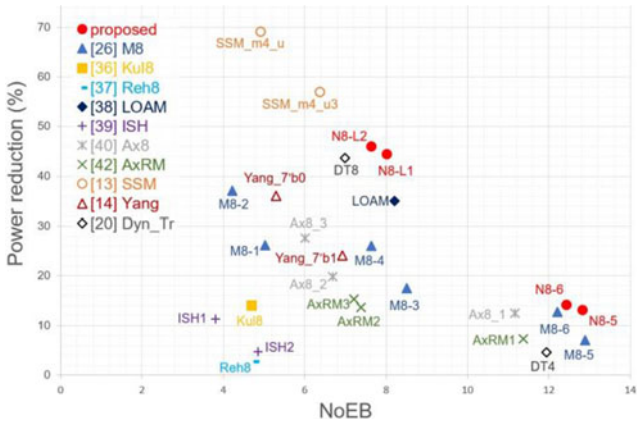

FIGURE 10. Power reduction of the considered 8×8 Approximate Multipliers with respect to the exact one vs Number of Effective Bits. The proposed circuits have lower power for the same NoEB. The exact design would have NoEB = 16 and zero power reduction.

TABLE 4. Performances of 8×8 approximate multipliers.

Design		Area* [μm²]	Power @1GHz Total [μW]	Power @1GHz Reduction [%]	Min Delay* [ps]	Error Rate [%]	NMED	MRED	NoEB
	Exact	81.45	109.38	-	220	-	-	-	16
Proposed	N8-L1	45.86	60.84	44.38	181	76.86	2.6×10^{-3}	2.4×10^{-2}	8.01
	N8-L2	**43.79**	**59.11**	**45.96**	174	75.56	2.9×10^{-3}	2.9×10^{-2}	7.64
	N8-5	70.68	95.01	13.14	229	**35.94**	$\mathbf{6.1\times10^{-5}}$	$\mathbf{1.2\times10^{-3}}$	12.83
	N8-6	70.02	93.83	14.21	222	37.11	8.5×10^{-5}	1.6×10^{-3}	12.42
[26]	M8-1	57.43	80.75	26.18	195	72.56	1.7×10^{-2}	6.1×10^{-2}	5.03
	M8-2	47.56	68.83	37.08	**165**	72.58	2.8×10^{-2}	8.4×10^{-2}	4.22
	M8-3	64.46	90.26	17.48	205	65.63	1.8×10^{-3}	1.6×10^{-3}	8.51
	M8-4	57.52	80.85	26.09	186	65.80	3.1×10^{-3}	2.2×10^{-2}	7.63
	M8-5	72.38	101.61	7.10	225	**35.94**	$\mathbf{6.1\times10^{-5}}$	1.3×10^{-3}	**12.88**
	M8-6	70.94	95.45	12.73	226	**35.94**	9.6×10^{-5}	1.8×10^{-3}	12.20
[36]	Kul8	63.29	94.08	13.99	204	46.73	1.4×10^{-2}	3.3×10^{-2}	4.69
[37]	Reh8	84.27	106.31	2.81	240	81.44	2.1×10^{-2}	1.5×10^{-1}	4.76
[38]	LOAM	50.78	71.10	35.00	209	74.76	2.0×10^{-3}	1.8×10^{-2}	8.21
[39]	ISH1	66.22	97.03	11.29	203	46.73	2.3×10^{-2}	4.8×10^{-2}	3.80
	ISH2	76.84	104.18	4.76	216	42.65	1.2×10^{-2}	2.8×10^{-2}	4.85
[40]	Ax8_1	71.20	95.73	12.48	225	53.91	2.4×10^{-4}	4.7×10^{-3}	11.16
	Ax8_2	68.40	87.68	19.84	215	70.46	5.5×10^{-3}	3.9×10^{-2}	6.69
	Ax8_3	60.46	79.18	27.61	199	83.88	1.1×10^{-2}	7.4×10^{-2}	6.00
[42]	AxRM1	71.03	101.35	7.35	229	75.00	2.6×10^{-4}	7.7×10^{-3}	11.36
	AxRM2	65.20	94.43	13.67	205	89.36	4.3×10^{-3}	1.5×10^{-1}	7.38
	AxRM3	62.80	92.59	15.35	209	96.17	5.2×10^{-3}	2.1×10^{-1}	7.20
[13]	SSM_m4	28.42	33.73	69.16	165	97.85	2.7×10^{-2}	1.5×10^{-1}	4.91
	SSM_m4_u3	36.11	47.08	56.96	194	97.85	9.0×10^{-3}	6.5×10^{-2}	6.37
[14]	Yang_7′b0	59.70	69.96	36.04	161	80.02	1.6×10^{-2}	9.1×10^{-2}	5.29
	Yang_7′b1	59.70	82.93	24.18	161	36.16	2.5×10^{-3}	8.5×10^{-3}	6.93
[20]	DT0	93.47	115.8	-5.87	249	0.00	0	0	16
	DT2	93.47	113.62	-3.88	249	50.00	1.9×10^{-5}	7.7×10^{-4}	14.45
	DT4	93.47	104.41	4.55	249	81.25	1.9×10^{-4}	5.6×10^{-3}	11.93
	DT8	93.47	61.66	43.62	249	98.05	6.9×10^{-3}	9.8×10^{-2}	6.99

**Area and power are reported for the circuits synthesized with a timing constraint of 1000ps. min delay is the minimum timing at which the circuit can be synthesized with a non-negative slack.*

blocks. For the less accurate designs, Ax8_3 with one accurate module, manages to surpass M8-1 that uses no accurate modules, both in accuracy and in power reduction. However, it is slightly larger and slower.

An interesting architecture, is proposed in [38]. It uses one exact multiplier, two custom modules, and an OR-based 4×4 approximate multiplier for the least significant part. This relatively small design, in terms of accuracy performs similarly to the proposed design N8-L1, as well as to M8-3 and M8-4. It achieves a significant power reduction with respect to M8-3 and M8-4 but N8-L1 leads. Among circuits with a similar power reduction percentage, M8-2 and Yang_7'b1, it exhibits a far more accurate behavior.

As it can be seen in Table 4, among the recursive topologies, the proposed circuits N8-L1 and N8-L2 occupy the smallest area and achieve the biggest reduction in power consumption. Moreover, they are very fast circuits, bested only by multipliers proposed in [13] and [14], that are not recursive, but optimized for a given bit width. At the same time, they exhibit competitive behavior in terms of accuracy. As it can be observed in Figure 10, even though there are more precise circuits in the literature, N8-L1 and N8-L2 provide a certain level of accuracy at a very low cost.

On the other hand, proposals N8-5 and N8-6, are very accurate circuits, exploiting three exact, and one proposed 4×4 multipliers. They offer a very high number of effective bits, matched only by the designs, M8-5 and M8-6, [26]. However, exploiting the proposed designs of N1 and N2, N8-5 and N8-6, achieve a greater power reduction, as it can be observed in Figure 10 and Table 4.

As demonstrated, we have designed four 8×8 multipliers, two with a NoEB around 8, and two with a NoEB around 12, that to the best of our knowledge exhibit a significant advancement with respect to the state-of-the-art.

TABLE 5. 16×16 Approximate multiplier compositions.

Design		a_Hb_H	a_Lb_H	a_Hb_L	a_Lb_L
Proposed	N16-5	N8-5	N8-5	N8-5	N8-5
	N16-6	N8-6	N8-6	N8-6	N8-6
[26]	M16-1	M8-1	M8-1	M8-1	M8-1
	M16-2	M8-2	M8-2	M8-2	M8-2
	M16-3	M8-3	M8-3	M8-3	M8-3
	M16-4	M8-4	M8-4	M8-4	M8-4
	M16-5	M8-5	M8-5	M8-5	M8-5
	M16-6	M8-6	M8-6	M8-6	M8-6
[36]	Kul16	Kul8	Kul8	Kul8	Kul8
[37]	Reh16	Reh8	Reh8	Reh8	Reh8
[40]	Ax16_1	Ax8_1	Ax8_1	Ax8_1	Ax8_1
	Ax16_2	Ax8_2	Ax8_2	Ax8_2	Ax8_2
	Ax16_3	Ax8_3	Ax8_3	Ax8_3	Ax8_3
[42]	AxRM16_1	AxRM1	AxRM1	AxRM1	AxRM1
	AxRM16_2	AxRM2	AxRM2	AxRM2	AxRM2
	AxRM16_3	AxRM3	AxRM3	AxRM3	AxRM3
Proposed	N16-L1	Exact	N8-L1	N8-L1	OR-Based
	N16-L2	Exact	N8-L2	N8-L2	OR-Based
[38]	LOAM16	Exact	LOAM	LOAM	OR-Based

C. 16×16 APPROXIMATE MULTIPLIERS

The 8×8 designs and the methodologies described above, can be used to scale up to 16×16 multipliers. As already shown in section III, two different approaches are used to generate 16×16 designs. Table 5 summarizes the architectures of the considered designs. The circuits following the most straightforward approach (exact sub-product addition) are presented at the top part of table 5, while the ones using the technique presented in [38] (approximate sub-product addition), are at the bottom part.

The performances of 16×16 approximate recursive multipliers are shown in Table 6. All 16×16 designs have been synthesized under the same timing constraint: 1000ps. Furthermore, they have been simulated with the same set of 10^5 uniformly distributed random vectors, with an input switching frequency equal to 1GHz.

The architecture proposed in [38], results in designs that significantly outperform other contributions. It should be noted that the most straightforward approach from an algorithmic point of view, followed by the circuits at the top part of Table 5, does not result in optimal configurations. In fact, each 16×16 multiplier is composed by four identical approximate 8×8 multipliers, that in turn may be composed by some exact 4×4 multipliers. On the other hand, [38] employs a single exact 8×8 multiplier placed in the most significant part, thus making an important impact on accuracy, despite the final approximate addition.

Moreover, the non-recursive exact and OR-based 8×8 multipliers in the most and least significant parts respectively, as well as the approximation in the final addition, allow this architecture to exploit minimal hardware resources. Therefore, the three

TABLE 6. Performances of 16×16 approximate multipliers.

Design		Area* [μm²]	Power @1GHz Total [μW]	Power @1GHz Reduction [%]	Min Delay* [ps]	Error Rate [%]	NMED	MRED	NoEB
	Exact	356.30	677.87	-	335	-	-	-	32
Proposed	N16-L1	226.35	227.87	66.38	355	99.31	2.65×10^{-5}	7.44×10^{-4}	14.91
	N16-L2	**224.88**	**225.70**	**66.70**	346	99.34	3.27×10^{-5}	8.84×10^{-4}	14.61
	N16-5	377.04	505.09	25.49	407	72.57	6.12×10^{-5}	1.13×10^{-3}	12.99
	N16-6	357.09	459.20	32.26	405	73.67	8.48×10^{-5}	1.54×10^{-3}	12.54
[26]	M16-1	305.02	428.70	36.76	359	96.67	1.65×10^{-2}	6.10×10^{-2}	5.03
	M16-2	266.55	384.63	43.26	328	96.68	2.75×10^{-2}	8.44×10^{-2}	4.21
	M16-3	359.28	459.20	32.26	367	94.61	1.75×10^{-3}	1.53×10^{-2}	8.52
	M16-4	310.16	422.22	37.71	346	94.68	3.10×10^{-3}	2.21×10^{-2}	7.63
	M16-5	393.73	522.00	22.99	410	**72.55**	6.07×10^{-5}	1.17×10^{-3}	13.05
	M16-6	368.58	495.40	26.92	403	72.57	9.57×10^{-5}	1.68×10^{-3}	12.30
[36]	Kul16	351.65	515.66	23.93	391	81.05	1.38×10^{-2}	3.31×10^{-2}	4.69
[37]	Reh16	428.64	512.18	24.44	412	98.34	2.09×10^{-2}	1.44×10^{-1}	4.75
[38]	LOAM16	238.04	227.42	66.45	**297**	99.25	$\mathbf{1.43\times10^{-5}}$	$\mathbf{4.93\times10^{-4}}$	**15.76**
[40]	Ax16_1	378.60	496.87	26.70	404	88.6	2.40×10^{-4}	4.41×10^{-3}	11.22
	Ax16_2	352.17	459.50	32.21	393	95.18	5.48×10^{-3}	3.86×10^{-2}	6.69
	Ax16_3	319.13	400.57	40.91	357	99.16	1.09×10^{-2}	7.31×10^{-2}	5.99
[42]	AxRM16_1	395.49	547.76	19.19	401	97.85	2.57×10^{-4}	1.18×10^{-2}	11.42
	AxRM16_2	363.25	528.08	22.10	390	99.52	4.27×10^{-3}	2.15×10^{-1}	7.39
	AxRM16_3	350.54	518.46	23.52	391	99.97	5.19×10^{-3}	3.44×10^{-1}	7.20

**Area and power are reported for the circuits synthesized with a timing constraint of 1000ps. min delay is the minimum timing at which the circuit can be synthesized with a non-negative slack.*

TABLE 7. Performances of the proposed 32×32 approximate multipliers.

Design		Area* [μm²]	Power @1GHz Total [μW]	Power @1GHz Reduction [%]	Min Delay* [ps]	Error Rate [%]	NMED	MRED	NoEB
	Exact	1477.6	3122.1	-	448	-	-	-	64
Proposed	N32-L1	1074.1	1860.2	40.42	553	100.00	9.6×10^{-10}	4.9×10^{-8}	29.78
	N32-L2	1073.1	1973.2	36.80	540	100.00	1.1×10^{-9}	5.5×10^{-8}	29.53
	N32-5	1933.63	3875.9	-24.14	590	92.66	6.1×10^{-5}	1.1×10^{-3}	13.01
	N32-6	1932.22	3940.2	-26.20	590	92.97	8.4×10^{-5}	1.5×10^{-3}	12.54

**Area and power are reported for the circuits synthesized with a timing constraint of 1000ps. min delay is the minimum timing at which the circuit can be synthesized with a non-negative slack.*

designs that follow this approach are the smallest, fastest, and least-power hungry. The circuit proposed in [38], manages to outperform N16-L1 and N16-L2 in accuracy, while N16-L2 slightly overcomes LOAM in terms of power reduction. N16-L2 also occupies the smallest area. Among the strictly recursive designs, N16-5 and N16-6 achieve a higher power reduction than circuits with similar or even lower accuracy.

D. PROPOSED 32×32 APPROXIMATE MULTIPLIERS

The performances of the proposed 32×32 approximate multipliers are shown in Table 7. The circuits have been synthesized under a timing constraint of 1000ps and simulated with 10^6 uniformly distributed random vectors, and an input switching frequency equal to 1GHz.

V. APPLICATIONS

Image processing is one of the most commonly considered error resilient applications and many papers test the proposed circuits in this scenario. In this paper, two image processing applications are considered: image blurring and image sharpening. The applications provide a more in depth understanding of the applicability range of the proposed designs.

A. IMAGE SMOOTHING

In image processing, low pass filtering results in image smoothing which effectively removes the high spatial frequency noise from the image. The low-pass filter exploits a moving kernel that processes one pixel at a time and modifies it considering the pixels in proximity. The processing of each pixel requires a number of multiplications that depends on the size of the kernel. In fact, the value of the modified pixel is the weighted average of the neighboring pixels. Moreover, image blurring is an error tolerant application, as the human eye is not able to detect trivial details.

TABLE 8. Gaussian kernels.

Original			Modified		
0.095	0.118	0.095	97	121	97
0.118	0.148	0.118	121	151	121
0.095	0.118	0.095	97	121	97

The kernel considered for smoothing is a two dimensional, rotationally symmetric, 3×3 Gaussian low-pass filter, with a standard deviation equal to 1.5, as in [27]. The floating-point numbers of the kernel are multiplied by 2^{10} and then rounded. In this way, the kernel's values are appropriate for the considered 8-bit input multipliers. The original and modified kernels are shown in Table 8.

Image processing, exploiting the investigated multipliers, has been performed aiming to blur a test image. The obtained images are shown in Figure 11. The same processing has been also performed with exact multipliers to provide an effective comparison for all designs. The structural similarity index (SSIM) and the peak signal to noise ratio (PSNR) provide a numerical indication of each multiplier's performance in image smoothing.

The results are shown in Table 9. The recursive designs are in the top part of the table, while the non-recursive ones occupy the bottom part. The proposed circuits N8-5 and N8-6 share the best results with the designs M8-5 and M8-6 proposed in [26] and Ax8_1 proposed in [40]. N8-L1 and N8-L2 follow close behind but still show a competitive behavior while achieving the greatest power reduction, as shown in Figure 10.

B. IMAGE SHARPENING

Sharpening or high pass filtering aims to make fine details more distinct and remove the blurring of a digital image, by enhancing transitions in the spatial intensity of the image. High frequencies are boosted while low frequencies are reduced. It should be noted that over-sharpening might result in unwanted halo artifacts.

The image sharpening process is similar to the smoothing process, but it uses a different kernel for the convolution. The authors in [14], [19], [25], [35] presented an image sharpening application exploiting the following 5×5 kernel:

$$Mask = \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \quad (19)$$

FIGURE 11. **Gaussian smoothing of images obtained with different multipliers. The circuits proposed in this paper are highlighted in bold.**

The output pixels of the sharpened image are given by:

$$Y(i,j) = 2 \cdot X(i,j) - \frac{1}{273} \sum_{m=-2}^{2} \sum_{n=-2}^{2} [X(i+m, j+n) \times Mask(m+3, n+3)] \quad (20)$$

In (20), $X(i, j)$ denotes a pixel from the input image, while $Y(i, j)$ from the sharpened output.

We have used the considered approximate multipliers, as well as an exact multiplier to sharpen an RGB test image. The results are demonstrated in Figure 12. SSIM and PSNR with respect to the sharpened image by exact multipliers are reported in Table 10. All proposed circuits have a high similarity ratio with the reference image. Even though there are better performing multipliers for this application, the proposed circuits exhibit reasonable behavior for such low-power designs.

C. IMAGE CLASSIFICATION WITH CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNNs) play an increasingly important role in machine learning, particularly for image recognition, object identification and speech recognition tasks. CNNs are error-tolerant and require a huge number of multiplications, therefore they are ideal candidates for using approximate multipliers [4].

TABLE 9. Performances of 8×8 approximate multipliers in image smoothing.

Design		SSIM [%]	PSNR [dB]
Proposed	N8-L1	97.85	41.7
	N8-L2	97.66	39. 5
	N8-5	97.98	43.0
	N8-6	97.98	43.0
[26]	M8-1	90.86	28.8
	M8-2	90.18	23.9
	M8-3	97.93	42.2
	M8-4	97.72	40.7
	M8-5	97.98	43.0
	M8-6	97.98	43.0
[36]	Kul8	97.81	41.0
[37]	Reh8	78.88	18.5
[38]	LOAM	97.90	42.4
[39]	ISH1	97.38	39.8
	ISH2	97.88	42.0
[40]	Ax8_1	97.96	43.0
	Ax8_2	97.85	39.2
	Ax8_3	97.25	35.6
[42]	AxRM1	97.97	43.0
	AxRM2	97.90	41.5
	AxRM3	97.85	41.2
[13]	SSM_m4	94.39	26.8
	SSM_m4_u3	96.41	38.9
[14]	Yang_7′b0	93.44	29.1
	Yang_7′b1	97.44	38.9
[20]	DT2	97.67	42.31
	DT4	97.67	42.31
	DT8	97.37	35.61

We performed some experiments of image recognition with the investigated approximate multipliers, by using a simple CNN composed by 9 layers, not counting the input one. The CNN includes two convolutional layers, each one followed by batch normalization and Rectified Linear Unit (ReLU) layers, a max pooling layer, a fully connected layer a final softmax layer. Two datasets have been considered: MNIST and SVHN. The former is a dataset of handwritten digits containing 70000 28×28-pixel, greyscale images split into 60000 training images and 10000 testing images [44]. The Street View House Number (SVHN) dataset contains 100000 32×32 RGB images of house numbers obtained from Google Street View, divided in 73257 training and 26032 test images [45]. In our tests, SVHN images have been converted into greyscale as the color has no significance in the classification [4].

The training of the CNNs has been performed in Matlab, by using floating-point arithmetic. After training, quantization of the convolutional and fully connected layers, requiring the vast majority of calculations, has been performed, to allow the testing of the approximate multipliers. We use test images to exercise the network and collect the dynamic ranges of the inputs of convolutional and fully connected layers. These inputs are positive values, due to the ReLU layers, and are easily quantized as 8-bit unsigned numbers that can directly feed the multipliers. The weights in the convolutional and fully connected layers of the network, on the other hand, are learnt during training and are signed numbers. Therefore, following [43], after quantization we converted the weights in sign-magnitude representation to perform multiplications using the investigated unsigned approximate multipliers.

Classification results are reported in Table 11. Column "Acc. loss" refers to the reduction in classification accuracy (in percentage) compared to the floating-point multiplier.

For the MNIST dataset, the considered CNN in floating-point implementation shows a remarkable accuracy of more than 99%. The accuracy remains almost unchanged when using exact 8-bit multiplier after network quantization. The majority of the investigated approximate multipliers perform well with this simple dataset, with some exceptions (Yang_7'b0, Ax8_2, Ax8_3, SSM_m4, Kul8, Reh8, AxRM3, ISH1). The proposed N8_L1 gives very good results, showing a mere 0.64% reduction in accuracy, with more than 44% power saving.

For the SVHN dataset the CNN accuracy is about 87%. In this case, network quantization yields a slight accuracy improvement, a phenomenon already observed in literature [4].

Several approximate multipliers yield a large accuracy reduction in this more demanding application. The multipliers giving an accuracy drop lower than 0.5% are: proposed N8_5 and N8_6, DT2 of [20], M8_5 and M8_6 of [26], Ax8_1 of [40] and AxRM1 [42]. Among these, the proposed N8_6 gives the best power reduction of more than 14%. Design Yang_7'b1 of [14] also performs well, with a reduction in accuracy of 3.3% and a power saving of more than 24%.

VI. CONCLUSION

In this paper we introduced two low-energy 4×4 approximate multipliers, obtained by simplifying the sum and carry expressions of the partial product matrix adders. These designs were recursively used to generate 8×8, 16×16 and 32×32 approximate multipliers, following two different architectures. Two 8×8 designs have been proposed with a NoEB around 8, and two with a NoEB around 12. Each 8×8 approximate multiplier consists of exact, proposed and OR-based, 4×4 designs. By exploiting the low power proposed circuits, N1 and N2, our approximate multipliers achieve great power reduction while still exhibiting competitive error performance. The error vs. power trade-off is compared with state-of-the-art approximated multipliers showing an improvement for both architectures. The proposed 8×8 circuits are tested in image processing and image classification using convolutional neural network demonstrating that these can be used to save power without sacrificing the result in typical error resilient applications.

FIGURE 12. **Image sharpening obtained with different multipliers. The circuits proposed in this paper are highlighted in bold.**

TABLE 10. **Performances of 8×8 approximate multipliers in image sharpening.**

Design		SSIM [%]	PSNR [dB]	Design		SSIM [%]	PSNR [dB]
Proposed	N8-L1	99.47	38.2	[36]	Kul8	99.97	56.9
	N8-L2	99.41	37.4	[37]	Reh8	79.17	22.2
	N8-5	99.92	56.6	[38]	LOAM	99.53	44.0
	N8-6	99.88	53.9	[39]	ISH1	99.97	56.6
[26]	M8-1	99.77	48.9		ISH2	99.96	52.9
	M8-2	99.58	40.3	[40]	Ax8_1	99.85	53.9
	M8-3	99.77	48.9		Ax8_2	97.67	22.7
	M8-4	99.58	40.3		Ax8_3	97.67	22.7
	M8-5	99.96	60.7	[42]	AxRM1	99.59	49.3
	M8-6	99.88	54.0		AxRM2	98.32	26.4
[13]	SSM_m4	93.19	18.5		AxRM3	80.78	28.3
	SSM_m4_u3	96.52	31.2	[20]	DT2	99.95	59.3
[14]	Yang_7′b0	94.37	29.2		DT4	99.86	49.3
	Yang_7′b1	99.92	51.3		DT8	96.51	23.8

TABLE 11. Image classification results using 8×8 approximate multipliers.

Design		MNIST		SVHN	
		Accuracy	Acc. loss	Accuracy	Acc. loss
	Floating Point	99.04%	–	87.18%	–
	8bit Exact	99.01%	0.03%	87.25%	-0.07%
Proposed	N8-L1	98.40%	0.64%	76.71%	10.48%
	N8-L2	97.43%	1.61%	70.82%	16.36%
	N8_5	99.01%	0.03%	87.01%	0.17%
	N8_6	99.00%	0.04%	86.88%	0.30%
[26]	M8-1	98.46%	0.58%	50.93%	36.25%
	M8-2	98.48%	0.56%	40.44%	46.74%
	M8-3	98.92%	0.12%	83.54%	3.65%
	M8-4	98.06%	0.98%	75.93%	11.25%
	M8-5	99.00%	0.04%	87.08%	0.10%
	M8-6	99.01%	0.03%	86.88%	0.30%
[36]	Kul8	89.62%	9.42%	77.56%	9.62%
[37]	Reh8	77.12%	21.92%	24.58%	62.60%
[38]	LOAM	99.02%	0.02%	83.66%	3.52%
[39]	ISH1	68.75%	30.29%	77.67%	9.52%
	ISH2	96.40%	2.64%	73.99%	13.20%
[40]	Ax8_1	99.02%	0.02%	86.90%	0.28%
	Ax8_2	68.47%	30.57%	22.12%	65.06%
	Ax8_3	58.32%	40.72%	24.17%	63.01%
[42]	AxRM1	99.00%	0.04%	86.56%	0.62%
	AxRM2	96.98%	2.06%	56.20%	30.98%
	AxRM3	49.50%	49.54%	20.77%	66.41%
[13]	SSM_m4	17.65%	81.39%	18.20%	68.98%
	SSM_m4_u3	96.75%	2.29%	53.63%	33.55%
[14]	Yang_7′b0	75.09%	23.95%	36.65%	50.53%
	Yang_7′b1	98.68%	0.36%	83.84%	3.35%
[20]	DT2	99.02%	0.02%	87.17%	0.01%
	DT4	98.97%	0.07%	86.26%	0.92%
	DT8	72.17%	26.87%	25.32%	61.86%

REFERENCES

[1] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Des. Test*, vol. 33, no. 1, pp. 8–22, Feb. 2016, doi: 10.1109/MDAT.2015.2505723.

[2] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proc. 50th ACM/EDAC/IEEE Des. Automat. Conf.*, 2013, pp. 1–9, doi: 10.1145/2463209.2488873.

[3] K. Chen, P. Yin, W. Liu, and F. Lombardi, "A survey of approximate arithmetic circuits and blocks," *Inf. Technol.*, vol. 64, no. 3, pp. 79–87, 2022, doi: 10.1515/itit-2021-0055.

[4] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, "Improving the accuracy and hardware efficiency of neural networks using approximate multipliers," *IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, vol. 28, no. 2, pp. 317–328, Feb. 2020, doi: 10.1109/TVLSI.2019.2940943.

[5] L. B. Soares, M. M. A. da Rosa, C. M. Diniz, E. A. C. da Costa, and S. Bampi, "Design methodology to explore hybrid approximate adders for energy-efficient image and video processing accelerators," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 66, no. 6, pp. 2137–2150, Jun. 2019, doi: 10.1109/TCSI.2019.2892588.

[6] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 32, no. 1, pp. 124–137, Jan. 2013, doi: 10.1109/TCAD.2012.2217962.

[7] L. Chen, J. Han, W. Liu, P. Montuschi, and F. Lombardi, "Design, evaluation and application of approximate high-radix dividers," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 4, no. 3, pp. 299–312, Jul.–Sep. 2018, doi: 10.1109/TMSCS.2018.2817608.

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

[9] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proc. IEEE*, vol. 108, no. 12, pp. 2108–2135, Dec. 2020, doi: 10.1109/JPROC.2020.3006451.

[10] 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, doi: 10.1109/TCSI.2018.2792902.

[11] 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, doi: 10.1109/TC.2018.2880742.

[12] U. Lotrič, R. Pilipović, and P. Bulić, "A hybrid radix-4 and approximate logarithmic multiplier for energy efficient image processing," *Electron. Low-Size Low-Power Sensors Syst.: From Custom Des. Embedded Solutions*, vol. 10, no. 10, May 2021, Art. no. 1175, doi: 10.3390/electronics10101175.

[13] A. G. M. Strollo, E. Napoli, D. De Caro, N. Petra, G. Saggese, and G. Di Meo, "Approximate multipliers using static segmentation: Error analysis and improvements," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 69, no. 6, pp. 2449–2462, Jun. 2022, doi: 10.1109/TCSI.2022.3152921.

[14] T. Yang, T. Ukezono, and T. Sato, "A low-power high-speed accuracy-controllable approximate multiplier design," in *Proc. 23rd Asia South Pacific Des. Automat. Conf.*, 2018, pp. 605–610, doi: 10.1109/ASPDAC.2018.8297389.

[15] M. Češka, J. Matyáš, V. Mrazek, L. Sekanina, Z. Vasicek, and T. Vojnar, "ADAC: Automated design of approximate circuits," in *Computer Aided Verification*. Berlin, Germany: Springer, 2018, doi: 10.1007/978-3-319-96145-3_35.

[16] S. Ullah, S. S. Murthy, and A. Kumar, "SMApproxLib: Library of FPGA-based approximate multipliers," in *Proc. 55th ACM/ESDA/IEEE Des. Automat. Conf.*, 2018, pp. 1–6, doi: 10.1109/DAC.2018.8465845.

[17] V. Mrazek, L. Sekanina, and Z. Vasicek, "Libraries of approximate circuits: Automated design and application in CNN accelerators," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 10, no. 4, pp. 406–418, Dec. 2020, doi: 10.1109/JETCAS.2020.3032495.

[18] P. Balasubramanian, R. Nayar, O. Min, and D. L. Maskell, "Approximator: A software tool for automatic generation of approximate arithmetic circuits," *Computers*, vol. 11, no. 1, Jan. 2022, Art. no. 11, doi: 10.3390/computers11010011.

[19] 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 Integration (VLSI) Syst.*, vol. 27, no. 5, pp. 1161–1173, May 2019, doi: 10.1109/TVLSI.2018.2890712.

[20] F. Frustaci, S. Perri, P. Corsonello, and M. Alioto, "Approximate multipliers with dynamic truncation for energy reduction via graceful quality degradation," *IEEE Trans. Circuits Syst. II: Exp. Briefs*, vol. 67, no. 12, pp. 3427–3431, Dec. 2020, doi: 10.1109/TCSII.2020.2999131.

[21] D. R. Kelly, B. J. Phillips, and S. Al-Sarawi, "Approximate signed binary integer multipliers for arithmetic data value speculation," *Proc. DASIP Conf.*, 2009, pp. 97–104.

[22] A. Cilardo, D. De Caro, N. Petra, F. Caserta, N. Mazzocca, and E. Napoli, "High speed speculative multipliers based on speculative carry-save tree," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 61, no. 12, pp. 3426–3435, Dec. 2014, doi: 10.1109/TCSI.2014.2337231.

[23] I. Qiqieh, R. Shafik, G. Tarawneh, D. Sokolov, and A. Yakovlev, "Energy-efficient approximate multiplier design using bit significance-driven logic compression," in *Proc. Des. Automat. Test Europe Conf. Exhib.*, 2017, pp. 7–12, doi: 10.23919/DATE.2017.7926950.

[24] D. Esposito, A. G. M. Strollo, and M. Alioto, "Low-power approximate MAC unit," in *Proc. 13th Conf. Ph.D. Res. Microelectronics Electron.*, 2017, pp. 81–84, doi: 10.1109/PRIME.2017.7974112.

[25] Y. Guo, H. Sun, L. Guo, and S. Kimura, "Low-cost approximate multiplier design using probability-driven inexact compressors," in *Proc. IEEE Asia Pacific Conf. Circuits Syst.*, 2018, pp. 291–294, doi: 10.1109/APCCAS.2018.8605570.

[26] 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, doi: 10.1109/JETCAS.2018.2832204.

[27] 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, doi: 10.1109/TCSI.2018.2839266.

[28] L. Maddisetty, R. K. Senapati, and J. V. R. Ravindra, "Training neural network as approximate 4:2 compressor applying machine learning algorithms for accuracy comparison," *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 8, no. 2, pp. 211–215, 2019, doi: 10.30534/ijatcse/2019/17822019.

[29] 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, doi: 10.1109/TCSI.2019.2918241.

[30] P. J. Edavoor, S. Raveendran, and A. D. Rahulkar, "Approximate multiplier design using novel dual-stage 4:2 compressors," *IEEE Access*, vol. 8, pp. 48337–48351, 2020, doi: 10.1109/ACCESS.2020.2978773.

[31] A. G. M. Strollo, E. Napoli, D. De Caro, N. Petra, and G. D. Meo, "Comparison and extension of approximate 4-2 compressors for low-power approximate multipliers," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 67, no. 9, pp. 3021–3034, Sep. 2020, doi: 10.1109/TCSI.2020.2988353.

[32] A. G. M. Strollo, D. De Caro, E. Napoli, N. Petra, and G. Di Meo, "Low-power approximate multiplier with error recovery using a new approximate 4-2 compressor," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2020, pp. 1–4, doi: 10.1109/ISCAS45731.2020.9180767.

[33] P. Zakian and R. N. Asli, "An efficient design of low-power and high-speed approximate compressor in FinFET technology," *Comput. Elect. Eng.*, vol. 86, Sep. 2020, Art. no. 106651, doi: 10.1016/j.compeleceng.2020.106651.

[34] H. Pei, X. Yi, H. Zhou, and Y. He, "Design of ultra-low power consumption approximate 4–2 compressors based on the compensation characteristic," *IEEE Trans. Circuits Syst. II: Exp. Briefs*, vol. 68, no. 1, pp. 461–465, Jan. 2021, doi: 10.1109/TCSII.2020.3004929.

[35] M. Khaleqi, M. Ahmadinejad, and M. H. Moaiyeri, "Ultraefficient imprecise multipliers based on innovative 4:2 approximate compressors," *Int. J. Circuit Theory Appl.*, vol. 49, no. 1, pp. 169–184, 2021, doi: 10.1002/cta.2876.

[36] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *Proc. 24th Int. Conf. VLSI Des.*, 2011, pp. 346–351, doi: 10.1109/VLSID.2011.51.

[37] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, J. Henkel, and J. Henkel, "Architectural-space exploration of approximate multipliers," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2016, pp. 1–8, doi: 10.1145/2966986.2967005.

[38] Y. Guo, H. Sun, and S. Kimura, "Design of power and area efficient lower-part-OR approximate multiplier," in *Proc. IEEE Region 10 Conf.*, 2018, pp. 2110–2115, doi: 10.1109/TENCON.2018.8650108.

[39] G. A. Gillani, M. A. Hanif, B. Verstoep, S. H. Gerez, M. Shafique, and A. B. J. Kokkeler, "MACISH: Designing approximate MAC accelerators with internal-self-healing," *IEEE Access*, vol. 7, pp. 77142–77160, 2019, doi: 10.1109/ACCESS.2019.2920335.

[40] H. Waris, C. Wang, W. Liu, J. Han, and F. Lombardi, "Hybrid partial product-based high-performance approximate recursive multipliers," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 1, pp. 507–513, Jan.–Mar. 2020, doi: 10.1109/TETC.2020.3013977.

[41] Z. Yang, X. Li, and J. Yang, "Power efficient and high-accuracy approximate multiplier with error correction," *J. Circuits Syst. Comput.*, vol. 29, no. 15, Dec. 2020, Art. no. 2050241, doi: 10.1142/S0218126620502412.

[42] H. Waris, C. Wang, C. Xu, and W. Liu, "AxRMs: Approximate recursive multipliers using high-performance building blocks," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 2, pp. 1229–1235, Apr.–Jun. 2022, doi: 10.1109/TETC.2021.3096515.

[43] M. Ahmadinejad and M. H. Moaiyeri, "Energy-and quality-efficient approximate multipliers for neural network and image processing applications," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 2, pp. 1105–1116, Apr.–Jun. 2022, doi: 10.1109/TETC.2021.3072666.

[44] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist

[45] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2011, Art. no. 5. [Online]. Available: http://ufldl.stanford.edu/housenumbers

EFSTRATIOS ZACHARELOS received the BS degree in physics from Aristotle University, Thessaloniki, Greece, in 2016, and the MS degree in electronic physics from Aristotle University, Thessaloniki, Greece, in 2019. He is currently working toward the PhD degree in electronic engineering with Federico II University, Naples, Italy. He is the co-author of four papers and his main research interests include real-time resampling, signal processing, and approximate computing.

ITALO NUNZIATA received the BS degree in electronic engineering from the University of Napoli Federico II, Italy. He is currently working toward the MS degree in electronic engineering with Federico II University, Italy, and the double degree in electronic and telecommunication with the University of Technology of Lodz, Poland. His current research interests include design and analysis of digital VLSI circuits and approximate computing.

GERARDO SAGGESE received the MSc degree (Hons.) in electronic engineering from the University of Napoli Federico II, Italy, and the double degree in electronic and telecommunication from the University of Technology of Lodz, Poland, in 2020. He is currently working toward the PhD degree in information technology and electrical engineering with the University of Napoli Federico II. His current research interests include signal processing, low power integrated circuit, brain machine interface and power circuits.

ANTONIO G. M. STROLLO (Senior Member, IEEE) received the MS degree (Hons.) and the PhD degree in electronic engineering from the University of Napoli Federico II, Italy. From 2002, he is a full professor with the University of Napoli Federico II, Italy, where he has been the head of the Department of Electronic and Telecommunication Engineering from 2005 to 2008. He has published more than 140 papers on international journals and conferences. His current research interests include design and analysis of digital VLSI circuits. Associate editor for *IEEE Transactions on Circuits and Systems-I* (2009 to 2012); currently he is associate editor for *Integration, the VLSI Journal*.

ETTORE NAPOLI (Senior Member, IEEE) received the PhD degree in electronic engineering in 1999, the electronic engineering degree (Hons.) in 1995, and the physics degree (Hons.) in 2009. He was a research associate with the Engineering Department, University of Cambridge, U.K., in 2004. Full professor with the University of Napoli Federico II in 2020. Full professor with the University of Salerno since 2021. He has authored or coauthored more than hundred articles published in international journals and conferences. His research interests include VLSI circuit design and modeling and design of power semiconductor devices.