Approximate Multipliers Using Static Segmentation: Error Analysis and Improvements

Antonio Giuseppe Maria Strollo, *Senior Member, IEEE*, Ettore Napoli, *Senior Member, IEEE*, Davide De Caro, *Senior Member, IEEE*, Nicola Petra, *Member, IEEE*, Gerardo Saggese, and Gennaro Di Meo

***Abstract*— Approximate multipliers are used in error-tolerant applications, sacrificing the accuracy of results to minimize power or delay. In this paper we investigate approximate multipliers using static segmentation. In these circuits a set of m contiguous bits (a segment of m bits) is extracted from each of the two n-bits operand, the two segments are in input to a small $m \times m$ internal multiplier whose output is suitably shifted to obtain the result. We investigate both signed and unsigned multipliers, and for the latter we propose a new segmentation approach. We also present simple and effective correction techniques that can significantly reduce the approximation error with reduced hardware costs. We perform a detailed comparison with previously proposed approximate multipliers, considering a hardware implementation in 28 nm technology. The comparison shows that static segmented multipliers with the proposed correction technique have the desirable characteristic of being on (or close to) the Pareto-optimal frontier for both power vs normalized mean error distance and power vs mean relative error distance trade-off plots. These multipliers, therefore, are promising candidates for applications where their error performance is acceptable. This is confirmed by the results obtained for image processing and image classification applications.**

***Index Terms*— Approximate computing, approximate multipliers, digital arithmetic.**

I. Introduction

THE request of energy-efficient and high-performance computing systems is steadily increasing, given the explosion of ubiquitous computing devices and the amount of data to be processed. Edge computing, for example, requires fast data analysis near data acquisition points to improve response times and save bandwidth, with the stringent power consumption constraints typical of battery-powered embedded and mobile systems.

The demand for low-power and high-speed circuits has driven the birth of new design paradigms. The Approximate Computing is emerging as a promising technique that achieve these goals by sacrificing the arithmetic accuracy of the results, with applications ranging from multimedia processing to machine learning [1]–[3].

In recent years, approximate arithmetic circuits (especially adders and multipliers) have been the subject of numerous investigations as they are often the most energy-hungry digital blocks. A comprehensive literature survey is given in [4].

Several approaches have been proposed to obtain highly efficient approximate multipliers. In logarithmic multipliers the product is obtained by summing an approximation of the logarithm of the operands and by calculating the antilogarithm of the sum [5]–[7]. Recursive multipliers use small elementary approximate multiplier blocks, suitably assembled, with typical block size 2×2 [9]–[11] or 4×4 [12]. Multiplier based on approximate compressors/counters to sum the partial products are investigated in [14]–[28], while [29]–[32] investigate truncated multipliers where some of the less-significant partial products are not formed and the resulting truncation error is mitigated with the help of suitable correction functions. The use of genetic optimization algorithms for the design space exploration of approximate multipliers is proposed in [33],[34], while approximate Booth multipliers are presented in [35],[36].

Customized approximate multipliers for efficient implementation of multiply-accumulate (MAC) blocks are presented in [11],[37] by exploiting forms of error mitigation and correction, in [38] by merging multiplier and an adder in an unique approximate structure, and in [39] by using approximate counters and deleting selected columns of the partial product matrix, while [40] presents a specialized square-accumulate block.

An important class of approximate multipliers, dubbed segmented multipliers, extracts a set of m contiguous bits (an m-bit segment) from each of the two n-bit operands, uses an inner $m \times m$ multiplier to multiply the two segments, and expands the $2m$-bit multiplication to the final $2n$-bit result. In multipliers using dynamic segmentation [41]–[45], the segments start from the leading one of the two operands and different levels of precision are obtained by truncating/rounding the segments and/or by approximating the $m \times m$ multiplier. This versatility is paid with a relatively complex hardware architecture, that includes leading-one detectors and shifters in addition to the (possibly approximate) $m \times m$ multiplier. In a Static Segmented Multiplier (SSM) [46] the operands are statically split into two m-bit segments. This approach greatly simplifies hardware implementation, reducing power consumption, since leading-one detectors and shifters are not required. On the other hand, for the same inner $m \times m$

Manuscript received November 10, 2021; revised January 28, 2022; accepted February 12, 2022. Date of publication March 3, 2022; date of current version May 27, 2022. This article was recommended by Associate Editor J. Di. *(Corresponding author: Antonio Giuseppe Maria Strollo.)*

Antonio Giuseppe Maria Strollo, Davide De Caro, Nicola Petra, Gerardo Saggese, and Gennaro Di Meo are with the Department of Electrical Engineering and Information Technology, University of Napoli Federico II, 80125 Napoli, Italy (e-mail: antonio.strollo@unina.it).

Ettore Napoli is with the Department of Information and Electrical Engineering and Applied Mathematics, University of Salerno, 84084 Fisciano, Italy.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2022.3152921.

Digital Object Identifier 10.1109/TCSI.2022.3152921

multiplier, static segmentation results in larger errors compared to dynamic segmentation. A hybrid approach is proposed in [47], using a first static segmentation stage and an inner multiplier employing dynamic segmentation.

In this paper we perform a detailed analysis of SSMs and we propose some improvements to the basic architecture. The main contributions can be summarized as follows:

i) We analyze the error characteristics of SSMs and, on the basis of this analysis, we propose a simple and effective error correction technique, able to significantly reduce the approximation error in unsigned SSMs, at reduced hardware cost.

ii) We develop a novel segmentation technique for signed SSMs.

iii) We propose a simple and hardware efficient correction technique for signed multiplier.

iv) We perform a detailed comparison with previously proposed, state of the art approximate multipliers and with signed SSMs obtained by transforming the operands in sign-modulus representation, multiplying the modules of the operand with an unsigned SSM and transforming the results in 2's complements representation.

To the best of our knowledge, no signed static segmented multipliers have been investigated so far, while the only paper to date that tries to improve the accuracy of static segmented multipliers is [48] that, however, requires the introduction of two carry-propagate adders before the $m \times m$ inner multiplier, with significant increase of delay and power

We have synthesized the circuits developed in this paper and several previously proposed approximate multipliers, using a commercial 28nm library, for 8bit operands. Syntheses show that our circuits, compared to previously prosed approximate multipliers, provide very good error-electrical performance trade-off. We have also investigated the performance of approximate multiplier in image filtering and image recognition applications. Obtained results confirm that SSMs are good candidates in applications where their error performance is acceptable.

The paper is organized as follows. The error metrics used throughout the paper are introduced in section II. The unsigned SSM is investigated in section III, where its error characteristics are derived, and the proposed error correction circuit is described. The section IV investigates the signed static segmented multipliers. Hardware implementation details are given in section V, where a comparison with previously proposed approximate multipliers is also reported. The section VI shows some application examples of proposed multiplier.

II. Error Metrics for Approximate Multipliers

Several metrics have been proposed to assess the error characteristics of approximate arithmetic circuits. Let us consider a $n \times n$ multiplier, with operands A and B, and let us indicate as $Y = A \times B$ the exact product and as Y' the approximate result. The error D, the error distance ED, and the relative error distance RED are defined as follows:

$$\begin{aligned} D &= Y - Y' \\ ED &= \left|Y - Y'\right| \\ RED &= \frac{ED}{|Y|} \quad \text{(for: } Y \neq 0\text{)} \end{aligned} \tag{1}$$

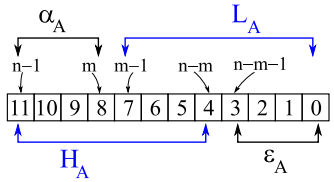

Fig. 1. Unsigned multiplier operand segmentation for $n = 12$, $m = 8$.

Let us also define the mean error μ_D and the mean square error D_{ms} as the average values of D and D^2, respectively. The Maximum Absolute Value of the accurate multiplier is indicated as MAV (for unsigned multipliers $MAV = (2^n - 1)^2$, while for signed multipliers $MAV = 2^{2n-2}$).

The metrics considered in the following are:

- *NM*: The Normalized Mean error, defined as μ_D/MAV
- *NmaxED*: The Normalized maximum Error Distance, defined as the maximum value of *ED* divided by *MAV*.
- *NMED*: The Normalized Mean Error Distance, defined as the average value of *ED* divided by *MAV*.
- *MRED*: The Mean Relative Error Distance, defined as the average value of *RED*
- *NoEB*: The Number of Effective Bits, defined in [16] as: $NoEB = 2n - \log_2\left(1 + \sqrt{D_{ms}}\right)$
- *PRED* introduced in [32] as the probability of having *RED* higher than 2 percent.

III. Unsigned Static Segmented Multipliers

A. Approximation Error in Unsigned SSM

The Fig. 1 shows the segmentation for the operand A, assuming $n = 12$. The m most significand bits of A, with $m \geq n/2$, represent the segment H_A, while the m less significant bits form the segment L_A ($m = 8$ in the example of Fig.1). In SSM, the operand A is approximated as follows:

$$A' = \begin{cases} H_A 2^{n-m} \text{ if: } \alpha_A \neq 0 \\ L_A \text{ if: } \alpha_A = 0 \end{cases} \tag{2}$$

where α_A comprises the n-m most significant bits of A. The approximation error using (2) is given by:

$$A - A' = \begin{cases} \varepsilon_A \text{ if: } \alpha_H \neq 0 \\ 0 \text{ if: } \alpha_H = 0 \end{cases} \tag{3}$$

where ε_A includes the n-m less significant bits of A. A similar segmentation is performed for B. The multiplication $A \times B$ is approximated in four different ways, depending on the position of the leading one in the two operands:

$$Y' = \begin{cases} H_A \times H_B 2^{2n-2m} & \text{if: } \alpha_A \neq 0;\ \alpha_B \neq 0 \\ H_A \times L_B 2^{n-m} & \text{if: } \alpha_A \neq 0;\ \alpha_B = 0 \\ H_B \times L_A 2^{n-m} & \text{if: } \alpha_A = 0;\ \alpha_B \neq 0 \\ L_A \times L_B & \text{if: } \alpha_A = 0;\ \alpha_B = 0 \end{cases} \tag{4}$$

The approximation error is given by:

$$D = \begin{cases} \left(\varepsilon_A H_B + \varepsilon_B H_A\right) 2^{n-m} + \varepsilon_A \varepsilon_B & (\alpha_A, \alpha_B \neq 0) \\ \varepsilon_A L_B & (\alpha_A \neq 0;\ \alpha_B = 0) \\ \varepsilon_B L_A & (\alpha_A = 0;\ \alpha_B \neq 0) \\ 0 & (\alpha_A,\ \alpha_B = 0) \end{cases} \tag{5}$$

TABLE I
ERROR CHARACTERISTICS OF UNSIGNED STATIC SEGMENTED MULTIPLIER

n	m	*Basic Architecture*						*Proposed Error Correction*					
		NM	*NMED*	*NoEB*	*MRED*	*NmaxED*	*PRED*	*NM*	*NMED*	*NoEB*	*MRED*	*NmaxED*	*PRED*
8	4	2.68×10^{-2}	2.68×10^{-2}	4.91	1.49×10^{-1}	1.14×10^{-1}	9.54×10^{-1}	7.62×10^{-3}	1.02×10^{-2}	6.21	7.56×10^{-2}	6.69×10^{-2}	7.58×10^{-1}
8	6	4.39×10^{-3}	4.39×10^{-3}	7.28	1.62×10^{-2}	2.34×10^{-2}	3.37×10^{-1}	7.01×10^{-4}	1.59×10^{-3}	8.76	8.05×10^{-3}	1.16×10^{-2}	8.19×10^{-2}
12	6	7.52×10^{-3}	7.52×10^{-3}	6.80	6.06×10^{-2}	3.05×10^{-2}	7.70×10^{-1}	1.64×10^{-3}	2.58×10^{-3}	8.22	2.69×10^{-2}	1.69×10^{-2}	3.20×10^{-1}
12	8	1.71×10^{-3}	1.71×10^{-3}	8.87	1.01×10^{-2}	7.31×10^{-3}	1.18×10^{-1}	3.25×10^{-4}	5.75×10^{-4}	10.3	4.22×10^{-3}	3.89×10^{-3}	2.33×10^{-2}
12	10	2.75×10^{-4}	2.75×10^{-4}	11.3	1.01×10^{-3}	1.46×10^{-3}	0	1.56×10^{-5}	1.00×10^{-4}	12.8	5.07×10^{-4}	6.10×10^{-4}	0
16	8	1.93×10^{-3}	1.93×10^{-3}	8.77	2.08×10^{-2}	7.77×10^{-3}	2.24×10^{-1}	4.45×10^{-4}	6.70×10^{-4}	10.2	9.49×10^{-3}	4.18×10^{-3}	7.62×10^{-2}
16	10	4.73×10^{-4}	4.73×10^{-4}	10.8	3.99×10^{-3}	1.92×10^{-3}	2.60×10^{-2}	1.06×10^{-4}	1.63×10^{-4}	12.2	1.73×10^{-3}	1.05×10^{-3}	6.42×10^{-3}
16	12	1.07×10^{-4}	1.07×10^{-4}	12.9	6.34×10^{-4}	4.58×10^{-4}	0	2.05×10^{-5}	3.60×10^{-5}	14.3	4.64×10^{-4}	2.43×10^{-4}	0

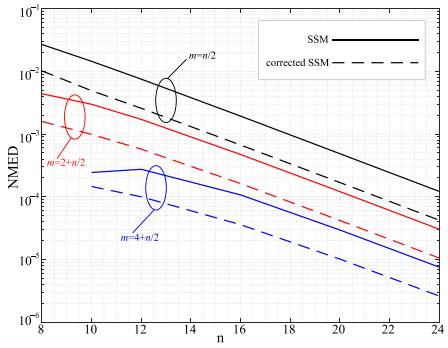

Fig. 2. Unsigned multiplier: NMED as a function of n, for the cases $m = n/2$, $m = 2 + n/2$, $m = 4 + n/2$. Solid lines: standard SSM; dashed lines: SSM with proposed error correction technique.

The last equation shows that the error D is always positive (i.e., Y' underestimates Y) and is largest in the first case, when both A and B are truncated to the m most-significant bits. In the second and third cases only one of the two operands is truncated, while the fourth case does not result in any error. The maximum error is attained when H_A and H_B reach the maximum value of $2^m - 1$ and ε_A and ε_B get the maximum value of $2^{n-m} - 1$. With simple algebra the normalized maximum error distance is computed as:

$$NmaxED = 2\frac{2^{n-m}-1}{2^n-1} - \frac{\left(2^{n-m}-1\right)^2}{\left(2^n-1\right)^2} \simeq 2\left(2^{-m}-2^{-n}\right) \tag{6}$$

where the last approximation holds for $n \gg 1$.The Table I, on the left, reports simulated error parameters for unsigned SSMs, obtained with 10 million random values for A and B. Note that $NMED = NM$, since the error is always positive. The Fig. 2 shows the behavior on *NMED* as a function of n, for the cases $m = n/2$, $m = 2+n/2$, $m = 4+n/2$. The *NMED* decreases exponentially with n. For a fixed n value, the *NMED* improves significantly by increasing m. A similar behavior is shown by MRED, albeit with a less sensible decrease with n.

B. Error Correction in Unsigned Multiplier

The equation (5) highlights that the error D is larger in the case $\alpha_A, \alpha_B \neq 0$, and is dominated by the term $(\varepsilon_A H_B + \varepsilon_B H_A)2^{n-m}$. To obtain a simple approximation for D when $\alpha_A, \alpha_B \neq 0$, let us approximate ε_A by rounding ε_A with its most significant digit:

$$\varepsilon_A \approx a_{n-m-1}2^{n-m-1} + 2^{n-m-2} = 2^{n-m-2}\left(2a_{n-m-1}+1\right) \tag{7}$$

A similar approximation is also performed for ε_B. The term H_A is given by:

$$H_A = \sum_{k=0}^{m-1} a_{k+n-m}2^k \tag{8}$$

A similar expression holds for H_B. By using (7)-(8) we obtain:

$$D \approx 2^{2n-2m-2}\sum_{k=0}^{m-1} 2^k c_k \tag{9}$$

where:

$$c_k = 2\left(a_{n-m-1}b_{k+n-m} + b_{n-m-1}a_{k+n-m}\right) + \left(b_{k+n-m} + a_{k+n-m}\right) \tag{10}$$

The value of c_k ranges from 0 to 6. To simplify the estimate of D, we approximate the coefficient c_k with a single bit, as:

$$c_k \simeq 4d_k \text{with:} d_k = \left(a_{n-m-1}b_{k+n-m}\right) OR \left(b_{n-m-1}a_{k+n-m}\right) \tag{11}$$

The Table II shows the values of c_k and of the approximation $4d_k$ for the various possible combinations of the bits at the right-hand side of (10). The maximum error in approximating c_k with $4d_k$ is 2, while the mean error is 1/4.

By substituting (11) in (9) we get:

$$D \approx 2^{2n-2m}\sum_{k=0}^{m-1} 2^k d_k \tag{12}$$

In practice, will use only a few most significant bits in (12) to simplify the hardware implementation. For instance, using three bits we have:

$$D* = d_{m-1}2^{2n-m-1} + d_{m-2}2^{2n-m-2} + d_{m-3}2^{2n-m-3} \tag{13}$$

Please note that the correction term $D*$ is very simple, comprising three terms each one given by the OR of two partial products.

In summary, we correct the SSM by modifying (4) as follows:

$$Y' = \begin{cases} H_A \times H_B 2^{2n-2m} + D* & \text{if: } \alpha_A \neq 0;\ \alpha_B \neq 0 \\ H_A \times L_B 2^{n-m} & \text{if: } \alpha_A \neq 0;\ \alpha_B = 0 \\ H_B \times L_A 2^{n-m} & \text{if: } \alpha_A = 0;\ \alpha_B \neq 0 \\ L_A \times L_B & \text{if: } \alpha_A = 0;\ \alpha_B = 0 \end{cases} \tag{14}$$

The Fig. 3 shows the partial product matrix of an $n = 8$, $m = 5$ static segmented multiplier, highlighting the partial products considered by the inner multiplier in the various cases.

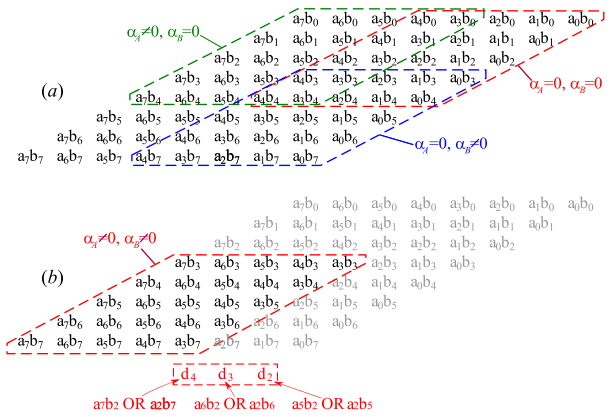

Fig. 3. Unsigned SSM partial product matrix and proposed error correction ($n = 8$, $m = 5$).

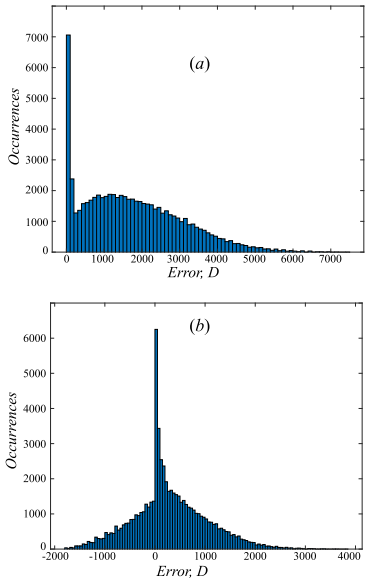

Fig. 4. Error histogram for an $n = 8$, $m = 4$ unsigned SSM. (a) Standard architecture (b) using proposed error correction.

In Fig. 3(a) the terms considered for $\alpha_A = 0$, $\alpha_B = 0$ are enclosed by the red line, those for $\alpha_A = 0$, $\alpha_B \neq 0$ by the blue line, and those for $\alpha_A \neq 0$, $\alpha_B = 0$ by the green line. The Fig. 3(b) refer to $\alpha_A \neq 0$, $\alpha_B \neq 0$. In this case the error is due to neglecting the greyed partial products. In the example of Fig. 3(b), the dropped terms with the largest weight are a_7b_2 and a_2b_7 (in general, they are $a_{n-1}b_{n-m-1}$ and $b_{n-1}a_{n-m-1}$). We mitigate the error by performing the OR of these two terms and including this value (corresponding to d_{m-1}in (13)) in the column on the left with respect to the original position of a_7b_2 and a_2b_7 (see Fig. 3(b)). By following similar considerations for the partial products a_6b_2, a_2b_6 and a_5b_2, a_2b_5 we obtain the three-bits error-correction term highlighted in red in Fig. 3.

The Fig. 4(a) shows the error histogram for an $n = 8$, $m = 4$ static segmented multiplier. The error is always positive; the cases corresponding to zero error ($\alpha_A = \alpha_B = 0$, (5)) are clearly visible. The Fig. 4(b) displays the error histogram with the proposed correction term. Comparison with Fig. 4(a) shows that the error is more uniformly distributed between negative and positive values, with a mean close to zero.

The error metrics for the segmented multiplier with the proposed error correction are also reported on the right side

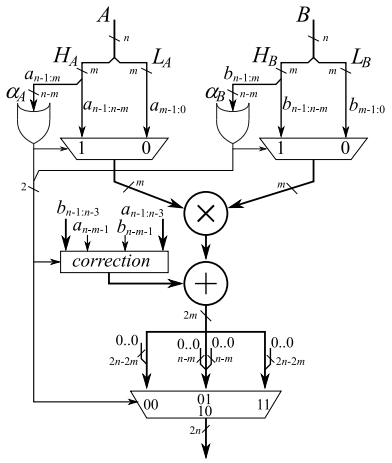

Fig. 5. Hardware architecture of unsigned SSM with error correction.

of Table I. Two-bits correction was employed for $n = 8$ and three-bits for other n values. The Fig. 2 shows with dashed lines the behavior on NMED as a function of n, with the proposed error correction technique. As it can be observed the error correction largely reduces the mean error, with an improvement of a factor of about 3 for *NMED*. The increase in *NoEB* of about 1.5 bit, while MRED and *NmaxED* are about halved.

The Fig. 5 shows the hardware architecture of the unsigned SSM with error correction. The circuit is very simple, with an inner multiplier-adder, that adds the correction term when $\alpha_A = \alpha_B = 1$.

IV. Signed Static Segmented Multipliers

A. Proposed Segmentation

For signed multiplier we could use the same approach of the previous sections, segmenting the operands as in Fig. 1 and approximating the result as in (4). In this case, however, the terms H_A and H_B are signed, while L_A and L_B are unsigned. Therefore, this straightforward implementation requires the use of a reconfigurable $m \times m$ inner multiplier, able to perform signed $\times$ signed, signed $\times$ unsigned, unsigned $\times$ signed and unsigned $\times$ unsigned multiplications, with a significant hardware overhead. A different approach is proposed in this paper. Let us consider the operand A (the same procedure applies to B) and let us rearrange A as shown in Fig. 6. The term L_A represents a signed number composed by the $m - 1$ less significant bits of A, augmented with the sign bit a_{n-1}.

With the proposed segmentation, when $\alpha_A = 0$ we have: $a_{n-1} = a_{n-2} = \cdots = a_{m-1} = 0$ and hence $A = L_A$. Similarly, for $\alpha_A = -1$ we get: $a_{n-1} = \cdots a_{m-1} = 1$ and again $A = L_A$ (note that L_A is a signed number). Therefore, we can approximate the operand A as follows:

$$A' \simeq \begin{cases} H_A 2^{n-m} & \text{if: } \alpha_A \notin \{0, -1\} \\ L_A & \text{if: } \alpha_A \in \{0, -1\} \end{cases} \tag{15}$$

where the approximation error is:

$$A - A' = \begin{cases} \varepsilon_A & \text{if: } \alpha_A \notin \{0, -1\} \\ 0 & \text{if: } \alpha_A \in \{0, -1\} \end{cases} \tag{16}$$

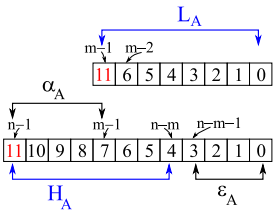

Fig. 6. Proposed signed multiplier operand segmentation for $n = 12$, $m = 8$.

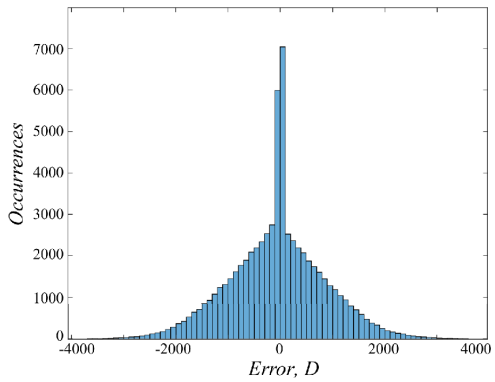

Fig. 7. Error histogram for an $n = 8$, $m = 4$ signed SSM.

We can repeat the same derivation for B, obtaining the following approximation for the product $A \times B$:

$$Y' = \begin{cases} H_A \times H_B 2^{2n-2m} & \text{if: } \alpha_A,\ \alpha_B \notin \{0,-1\} \\ H_A \times L_B 2^{n-m} & \text{if: } \alpha_A \notin \{0,-1\};\ \alpha_B \in \{0,-1\} \\ H_B \times L_A 2^{n-m} & \text{if: } \alpha_A \in \{0,-1\};\ \alpha_B \notin \{0,-1\} \\ L_A \times L_B & \text{if: } \alpha_A,\ \alpha_B \in \{0,-1\} \end{cases} \tag{17}$$

The implementation of the above equation requires a signed $m \times m$ multiplier (no reconfigurability needed). The approximation error is given by:

$$D = \begin{cases} (\varepsilon_A H_B + \varepsilon_B H_A)\, 2^{n-m} + \varepsilon_A \varepsilon_B; \\ \qquad \alpha_A,\ \alpha_B \notin \{0,-1\} \\ \varepsilon_A L_B \quad \alpha_A \notin \{0,1\};\ \alpha_B \in \{0,-1\} \\ \varepsilon_B L_A \quad \alpha_A \in \{0,1\};\ \alpha_B \notin \{0,-1\} \\ 0 \quad \alpha_A,\ \alpha_B \in \{0,-1\} \end{cases} \tag{18}$$

The equations (17)-(18) are similar to (4)(5), but there are two important differences: (a) the conditions to check for using one of the two input segments and (b) the signedness of both segments H and L (note also that α in Fig. 6 is one bit longer compared to Fig. 1).

B. Approximation Error in Signed SSM

The Fig. 7 shows the error histogram for an $n = 8$, $m = 4$ signed static segmented multiplier, using the segmentation proposed in this paper. The error (apart from the region where it is close to zero since α_A or $\alpha_B \in \{0,-1\}$) has a roughly triangular distribution, with a near-zero mean value. This can be explained by observing that the factor $(\varepsilon_A H_B + \varepsilon_B H_A)$ that dominates the error D in (18) includes the terms H_B and H_A than can be either negative or positive. Consequently, the error has a double-sided distribution and the mean error μ_D, by averaging negative and positive values, is much lower compared to unsigned SSM.

TABLE II

VALUES OF COEFFICIENTS IN (10) AND THEIR APPROXIMATION (11)

b_{k+n-m}	a_{k+n-m}	b_{n-m-1}	a_{n-m-1}	c_k	$4d_k$	$Error=c_k-4d_k$
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	1	0	1
0	1	0	1	1	0	1
0	1	1	0	3	4	-1
0	1	1	1	3	4	-1
1	0	0	0	1	0	1
1	0	0	1	3	4	-1
1	0	1	0	1	0	1
1	0	1	1	3	4	-1
1	1	0	0	2	0	2
1	1	0	1	4	4	0
1	1	1	0	4	4	0
1	1	1	1	6	4	2

The maximum error is attained when ε_A and ε_B get the maximum value of $2^{n-m}-1$. The error D reaches its maximum positive value when H_A and H_B are $2^{m-1} - 1$, while the minimum negative value of D is obtained for $H_A = H_B = -2^{m-1}$. The error distance is slightly larger in this second case, and the normalized maximum error distance can be expressed as:

$$\begin{aligned} NmaxED &= \frac{-2^n\left(2^{n-m}-1\right) + \left(2^{n-m}-1\right)^2}{2^{2n-2}} \\ &\simeq 4\left(2^{-m} - 2^{-n}\right) \end{aligned} \tag{19}$$

Thus, *NmaxED* roughly doubles compared to unsigned SSM. This is mainly due to the different Maximum Absolute Value of the multiplier (used to normalize *NmaxED*) that is about four times lower in signed multipliers.

The left side of Table III reports simulated error metrics for signed SSM, obtained with 10 million random values for A and B. Comparison with Table I shows that signed SSM has a much lower *NM*, while the *NMED* is larger (again, this is due to the different Maximum Absolute Value of the multiplier used to normalize *NMED*). The *NoEB* is better in signed multiplier, close to the one of unsigned multiplier with error correction, owing to the two-sided error distribution shown in Fig. 7. The *MRED* is higher in signed multipliers, compared to unsigned counterpart; this also can be attributed to factor $|Y|$ in the denominator of *RED* in (1), that tends to be lower in signed multiplier.

C. Error Correction in Signed Multiplier

The approximation error in signed SSM can be either positive or negative. Therefore, the correction factor must be added or subtracted from the multiplication result, depending on the sign of the error, and this makes the derivation of a correction strategy less straightforward, compared to unsigned SSM. A heuristic technique is proposed in this paper. As shown in the example of Fig. 6, the less-significant bit of segment H_A is given by a_{n-m}. We proposed to modify this less-significant bit as: a_{n-m} OR a_{n-m-1}, to partly compensate for the dropped

TABLE III
ERROR CHARACTERISTICS OF SIGNED STATIC SEGMENTED MULTIPLIER

		Proposed segmentation						*Error correction*					
n	*m*	*NM*	*NMED*	*NoEB*	*MRED*	*NmaxED*	*PRED*	*NM*	*NMED*	*NoEB*	*MRED*	*NmaxED*	*PRED*
8	4	-3.45×10^{-3}	4.05×10^{-2}	6.20	2.58×10^{-1}	2.21×10^{-1}	9.28×10^{-1}	-7.46×10^{-4}	3.06×10^{-2}	6.55	1.81×10^{-1}	2.19×10^{-1}	9.06×10^{-1}
8	6	-1.46×10^{-4}	6.66×10^{-3}	8.59	2.64×10^{-2}	4.63×10^{-2}	5.25×10^{-1}	-3.15×10^{-5}	5.56×10^{-3}	8.83	2.20×10^{-2}	4.60×10^{-2}	4.41×10^{-1}
12	6	-2.38×10^{-4}	1.12×10^{-2}	8.11	1.05×10^{-1}	6.06×10^{-2}	7.79×10^{-1}	-5.82×10^{-5}	8.54×10^{-3}	8.46	7.47×10^{-2}	6.05×10^{-2}	7.03×10^{-1}
12	8	-1.35×10^{-5}	2.53×10^{-3}	10.2	1.59×10^{-2}	1.46×10^{-2}	2.57×10^{-1}	-3.50×10^{-6}	1.97×10^{-3}	10.5	1.23×10^{-2}	1.46×10^{-2}	1.78×10^{-1}
12	10	-5.70×10^{-7}	4.16×10^{-4}	12.6	1.64×10^{-3}	2.93×10^{-3}	0	-3.07×10^{-7}	3.48×10^{-4}	12.8	1.37×10^{-3}	2.97×10^{-3}	0
16	8	-1.59×10^{-5}	2.87×10^{-3}	10.1	3.69×10^{-2}	1.55×10^{-2}	3.62×10^{-1}	-5.21×10^{-6}	2.20×10^{-3}	10.4	2.66×10^{-2}	1.55×10^{-2}	2.76×10^{-1}
16	10	-9.36×10^{-7}	6.99×10^{-4}	12.1	6.55×10^{-3}	3.84×10^{-3}	6.86×10^{-2}	-4.02×10^{-7}	5.38×10^{-4}	12.4	4.99×10^{-3}	3.84×10^{-3}	4.58×10^{-2}
16	12	-7.53×10^{-8}	1.58×10^{-4}	14.2	9.96×10^{-4}	9.15×10^{-4}	0	-5.94×10^{-8}	1.23×10^{-4}	14.5	7.71×10^{-4}	9.15×10^{-4}	0

TABLE IV
ERRORS IN APPROXIMATE SIGN-MODULUS CONVERSION

	modulus multiplication with exact multiplier								*modulus multiplication with SSM*					
n	*NM*	*NMED*	*NoEB*	*MRED*	*NmaxED*	*PRED*	*n*	*m*	*NM*	*NMED*	*NoEB*	*MRED*	*NmaxED*	*PRED*
8	4.58×10^{-5}	3.86×10^{-3}	9.56	4.13×10^{-2}	1.56×10^{-2}	4.15×10^{-1}	8	4	4.58×10^{-5}	1.11×10^{-2}	8.07	8.70×10^{-2}	7.42×10^{-2}	7.92×10^{-1}
							8	6	4.58×10^{-5}	4.16×10^{-3}	9.50	4.35×10^{-2}	1.93×10^{-2}	4.44×10^{-1}
12	1.79×10^{-7}	2.44×10^{-4}	13.56	3.99×10^{-3}	9.76×10^{-4}	2.51×10^{-2}	12	6	5.28×10^{-7}	2.60×10^{-3}	10.2	2.49×10^{-2}	1.73×10^{-2}	3.27×10^{-1}
							12	8	5.23×10^{-7}	6.26×10^{-4}	12.2	6.48×10^{-3}	4.38×10^{-3}	3.73×10^{-2}
							12	10	2.34×10^{-7}	2.56×10^{-4}	13.5	4.12×10^{-3}	1.10×10^{-3}	2.54×10^{-2}
16	5.60×10^{-10}	1.53×10^{-5}	17.56	3.31×10^{-4}	6.10×10^{-5}	1.50×10^{-3}	16	8	2.30×10^{-7}	6.71×10^{-4}	12.2	8.45×10^{-3}	4.38×10^{-3}	9.05×10^{-2}
							16	10	-4.59×10^{-8}	1.64×10^{-4}	14.2	1.67×10^{-3}	1.10×10^{-3}	3.77×10^{-3}
							16	12	1.53×10^{-8}	3.92×10^{-5}	16.2	4.83×10^{-4}	2.75×10^{-4}	1.52×10^{-3}

term ε_A. Note that the modification is irrelevant if $a_{n-m}=1$, while we add a correction term when $a_{n-m}=0$. In summary, we approximate A as follows:

$$A'' \simeq \begin{cases} H_A 2^{n-m} & \text{if: } \alpha_A \notin \{0,-1\}, \\ & a_{n-m}=1 \\ H_A 2^{n-m} + 2^{n-m} a_{n-m-1} & \text{if: } \alpha_A \notin \{0,-1\}, \\ & a_{n-m}=0 \\ L_A & \text{if: } \alpha_A \in \{0,-1\} \end{cases} \quad (20)$$

Let us focus on the case $\alpha_A \notin \{0,-1\}$ and let us assume that the bits of A have equal probability of being 0 or 1. By using the basic approximation (15), the error can be considered as a random variable uniformly distributed in: $[0,\ 2^{n-m}-1]$, and the mean approximation error is: $E[A-A']=(2^{n-m}-1)/2$.

By using the approximation (20), instead, the error is uniformly distributed in: $[0,\ 2^{n-m}-1]$ for $a_{n-m}=1$, whereas for $a_{n-m}=0$ it is uniformly distributed in: $[-2^{n-m-1},\ 2^{n-m-1}-1]$. Since these two cases are equally likely to occur, the mean approximation error can be easily calculated as: $E[A-A'']=(2^{n-m-1}-1)/2$, and is almost halved.

The proposed heuristic technique gives a minimal hardware overhead, while producing an improvement in most error metrics, as shown in Table II (on the right). The improvement for *NMED* and *MRED* is in the range 20%-25%, with a *NoEB* increase of about 0.3 bit. The maximum error distance remains almost unchanged.

D. Using Unsigned Multiplier With Sign-Modulus Conversion

Several approximate multipliers proposed in literature are unsigned only, and operation with signed operands is obtained in three steps: i) operands conversion to sign-modulus representation, ii) modulus multiplication using an inner unsigned multiplier, iii) conversion back of the result in 2s complements. The step i) is realized in an approximate way: if the sign bit of the operand is 1, its remaining $n-1$ bits are complemented and passed to the inner $(n-1)\times(n-1)$ multiplier. A similar technique is employed in step iii): if the sign bits of the two operands are different, the output of the inner multiplier are complemented, and the sign bit of the result is set to 1 [7], [8], [38], [43], [45], [49].

This approach has two drawbacks: it introduces additional errors (for using simple 1's complementation as opposed as 2's complementation) and it requires additional hardware to conditionally complement inputs and output. The left side of Table IV shows the error introduced by 1's complementation. The values in this Table have been obtained by simulating an ideal situation in which an exact multiplier is employed for the inner multiplication. Hence, the values in Table IV represent a lower-bound for the error achievable by using approximate sign-modulus conversion.

Despite their drawbacks, sign-modulus technique has also advantages, such as the use of a smaller $(n-1)\times(n-1)$ inner multiplier and the possibility of using well designed unsigned multiplier in application where signed operands are employed.

The error metrics for signed multipliers based on sign-modulus conversion (using as inner multiplier the SSM with the correction described in Section III-B) are reported on the right in Table IV. As it can be observed, error parameters are above the lower bounds of Table IV. The multipliers with sign-modulus conversion show sensible error reductions compared to the basic architecture, with a decrease of *NMED*, *MRED* and *NmaxED* in the range of 65%-70%, and a *NoEB* increase of about 2 bits, on average. This improvement is due to the use of a smaller inner multiplier that has moreover good accuracy owing to the error-correction technique.

V. HARDWARE IMPLEMENTATION

Previously proposed 8-bit approximate multipliers and the SSMs investigated in this paper have been described in HDL and synthesized in TSMC 28nm CMOS technology. Operating conditions are typical corner, standard-V_T devices, nominal

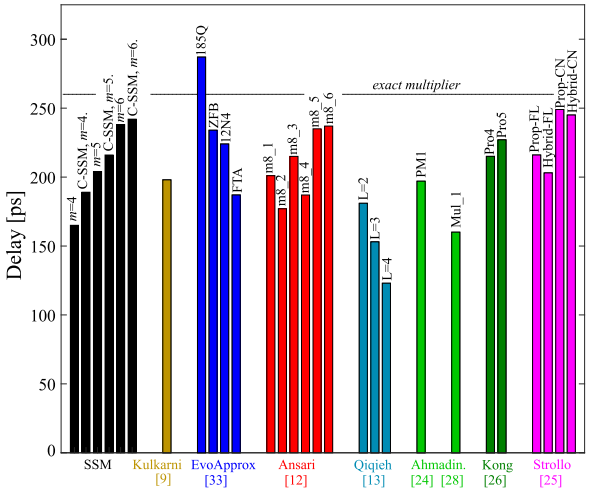

Fig. 8. Minimum delay of 8-bit approximate multipliers.

supply voltage of 0.9V, output loading corresponding to a fan-out of four inverters 2×. The syntheses have been performed with Cadence Genus, using physical synthesis to consider the parasitic effects of the interconnections. Power dissipation is computed by simulating the synthesized netlist with 100,000 random vectors, with a 1GHz toggle rate. To make a fair comparison, area and power have been obtained by imposing the same timing constraint of 500ps maximum delay to all circuits. To evaluate the maximum achievable speed, the circuits have also been synthesized with more stringent constraints, to obtain the minimum delay of each multiplier.

A. Unsigned Multipliers

Results for unsigned multipliers are summarized in Table V. This table reports also, for comparison, the performance of an exact Dadda multiplier and of several approximate multipliers presented in Literature: the multiplier proposed in [9] (indicated as *Kulkarni*); four multipliers selected from EvoApprox8b library [33] (named *mul8u_185Q* - *mul8u_FTA*), belonging to the Pareto-optimal subset in terms of Mean Absolute Error vs power or Mean Relative Error vs power; four versions the multipliers based on encoded partial products and approximate compressors proposed in [12] (named *m8_1 – m8_6*); three multiplier versions proposed in [13] using a significance-driven logic compression (named L = 2, L = 3, L = 4); the multipliers PM1 proposed in [28] and Mul_1 is [24], using partial products truncation and approximate 5-2 and 4-2 compressors, respectively; the Pro4 and Pro5 multipliers of [26] using approximate 4-2 compressors; two approximate multipliers of [25] (Proposed and Hybrid), in the two versions: FL, using approximate 4-2 compressors in all the partial-product matrix, and CN, using approximate 4-2 compressors only in the 8 left-most columns of the partial-product matrix. The Table V reports also the percentage variations of delay, area, and power with respect to the exact multiplier.

For static segmented multipliers the proposed error correction, being very simple, gives only a minor performance degradation (multipliers using proposed error correction are indicated as C-SSM in Table V and in the following Tables).

TABLE V
AREA, DELAY AND POWER OF EXACT AND APPROXIMATE 8 × 8 UNSIGNED MULTIPLIERS

	Delay (ps)	Delay reduct.	Area (μm^2)	Area reduct.	Power (μW)	Power reduct.
Exact	260	–	196	–	177	–
SSM, *m*=4	165	36.5%	66.5	66.1%	49.9	71.8%
C-SSM, *m*=4	189	27.3%	71.2	63.7%	54.5	69.2%
SSM, *m*=5	204	21.5%	101.5	48.2%	82.5	53.4%
C-SSM, *m*=5	216	16.9%	107.9	44.9%	92	48.0%
SSM, *m*=6	238	8.3%	143.5	26.8%	144.9	18.1%
C-SSM, *m*=6	242	6.9%	153.9	21.5%	147.8	16.5%
Kulkarni [9]	198	23.8%	158.4	19.2%	151.3	14.5%
mul8u_185Q [33]	287	-10.4%	113.4	42.1%	162.9	8.0%
mul8u_ZFB [33]	234	10.0%	145.5	25.8%	162.5	8.2%
mul8u_12N4 [33]	224	13.8%	90.9	53.6%	99.2	44.0%
mul8u_FTA [33]	187	28.1%	53.5	72.7%	66.1	62.7%
m8_1 [12]	201	22.7%	137	30.1%	127.5	28.0%
m8_2 [12]	177	31.9%	111.9	42.9%	99.85	43.6%
m8_3 [12]	215	17.3%	152.2	22.3%	138.3	21.9%
m8_4 [12]	187	28.1%	131.3	33.0%	117	33.9%
m8_5 [12]	235	9.6%	186.1	5.1%	165.7	6.4%
m8_6 [12]	237	8.8%	179.2	8.6%	160.6	9.3%
L=2 [13]	181	30.4%	121.7	37.9%	113.9	35.6%
L=3 [13]	153	41.2%	92.8	52.7%	90.1	49.1%
L=4 [13]	123	52.7%	70.9	63.8%	70.1	60.4%
PM1 [28]	197	24.2%	134.6	31.3%	139.4	21.2%
Mul_1 [24]	160	38.5%	82.8	57.8%	76.3	56.9%
Pro4 [26]	215	17.3%	179.2	8.6%	170.3	3.8%
Pro5 [26]	227	12.7%	180.9	7.7%	173.8	1.8%
Prop-FL [25]	216	16.9%	155.4	20.7%	140.8	20.5%
Hybrid-FL [25]	203	22.1%	126.2	35.6%	112.8	36.3%
Prop-CN [25]	249	4.4%	175.2	10.6%	162.0	8.5%
Hybrid-CN [25]	245	6.0%	163.4	16.6%	151.7	14.3%

For example, for $m = 5$, the power reduction compared to the exact multiplier is about 53% for the standard SSM multiplier and 48% for the SSM using the two-terms correction proposed in the paper. As shown in Table V, only the SSMs with $m = 4$ and $m = 5$, allow to reach a power reduction larger than 40% while the SSM with $m = 6$ reveals not very effective, with a power saving in the range 13-16%. The Kulkarni multiplier also offers a limited power reduction, while two of the selected EvoApprox8b multipliers (*mul8u_12N4* and *mul8u_FTA*) give a remarkable power saving. The various versions of multipliers proposed in [12], with different precision levels, allow to tailor the power saving from about 6% to more than 40%; a similar consideration applies to the architectures in [13] (that achieves more than 60% power reduction with the L=4 configuration). The multiplier proposed in [24] shows a remarkable power saving, close to 57%, while the architectures in [25] give a power saving ranging from less than 10% to more than 35%.

The area improvement is well correlated with power reduction in all the investigated circuits, except for EvoApprox8b *mul8u_185Q* and *mul8u_ZFB* that show a very good area reduction but limited power saving. This is probably due to glitching power. A similar trend is also observed for speed improvement, again with some exceptions, as shown in Fig. 8. The *mul8u_185Q* is slower than exact multiplier, whereas *mul8u_ZFB* and *mul8u_12N4*, while having significant differences in terms of power dissipation, show similar delays. The fastest multiplier is L=4 of [13].

Power reduction is often considered the main goal of approximate arithmetic circuits, so we have investigated in more detail the tradeoff between power saving and accuracy. The Fig. 9 shows the tradeoff curves (power reduction vs.

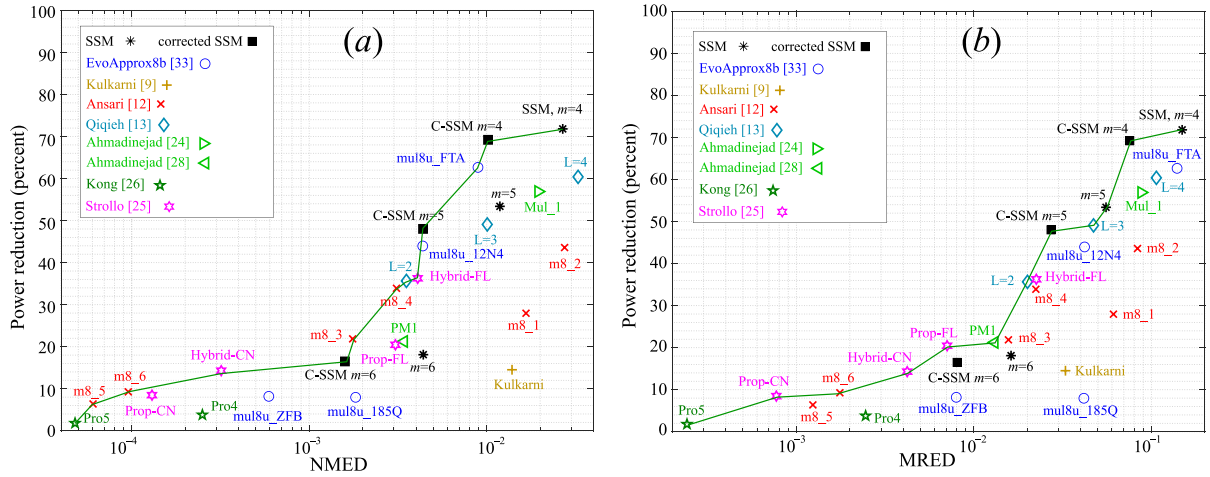

Fig. 9. Tradeoff between power saving and accuracy in unsigned multipliers. Each point corresponds to a multiplier, the closer the point to the top-left corner, the better the tradeoff. The green line is the Pareto-optimal frontier, given by multipliers with the best power-accuracy tradeoff. (*a*) Power reduction vs NMED; (*b*) Power reduction vs MRED.

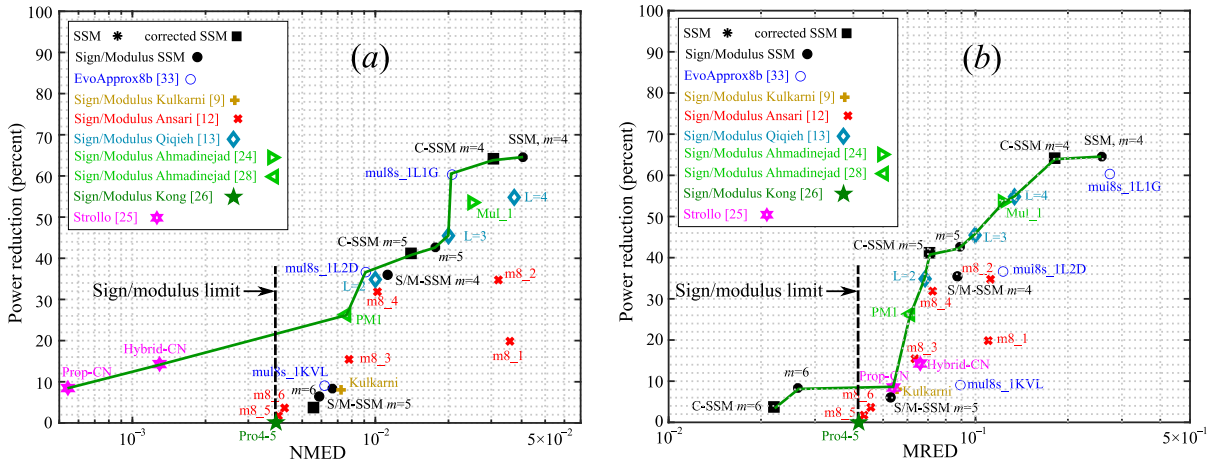

Fig. 10. Tradeoff between power saving and accuracy in signed multipliers. Each point corresponds to a multiplier, the closer the point to the top-left corner, the better the tradeoff. The green line is the Pareto-optimal frontier, given by multipliers with the best power-accuracy tradeoff. (*a*) Power reduction vs NMED; (*b*) Power reduction vs MRED.

NMED and MRED) for the investigated multipliers. Each symbol in this figure corresponds to a multiplier, the closer the symbol to the top-left corner, the better the tradeoff. The green line is the Pareto-optimal frontier, given by multipliers with the best power-accuracy tradeoff. Let us consider the power reduction vs NMED tradeoff, displayed in Fig. 9(a). The SSMs with $m = 4$ lie on the Pareto-optimal curve, as the SSMs with $m = 5$ and $m = 6$ using the proposed error correction technique. The EvoApprox8b multiplier *mul8u_FTA* and *mul8u_12N4* are on the Pareto curve, while the other two investigated EvoApprox8b multipliers are below the Pareto frontier. The Kulkarni multiplier in [9] and the multipliers in [28] and [24] are suboptimal. Four of the multipliers proposed in [12] are on the Pareto curve, with *m8_5* and *m8_6* showing a low NMED (but also a power reduction less than 10%). The lowest NMED is exhibited by Pro5 of [26] that, however, shows a negligible power saving. Of the multipliers proposed in [13], only the L = 2 is on the Pareto frontier. The hybrid architecture of [25] also performs well with both CN and FL configurations.

The power reduction-MRED tradeoff is shown in Fig. 9(b). The SSMs with $m = 4$ and $m = 5$ are on the Pareto frontier, while the $m = 6$ SSMs are below the curve. The four EvoApprox8b circuits are suboptimal in terms of MRED. The *m8_3*−*m8_6* multipliers of [12] remain close to the Pareto-frontier, while *m8_1* and *m8_2* are well below the optimal curve. Considering the circuits in [13], are on frontier both L = 2 and L = 3; the PM1 of [28] also shows a good MRED-power saving trade-off. The approximate multipliers in [25] perform well in terms of MRED, all being on (or close to) the Pareto-frontier.

The results displayed in Fig. 9 show that the SSMs with the proposed correction technique and $m = 4$ or $m = 5$ have the desirable characteristic of being on the Pareto-optimal frontier for both power vs NMED and power vs MRED tradeoff plots.

B. Signed Multipliers

Trade-off curves for signed multipliers are shown in Fig. 10. We report results for the signed SSMs investigated in this paper, in three versions: using the segmentation of section IV-A, the simplified error correction of section IV-C, and the sign-modulus conversion. We also considered three signed multipliers from EvoApprox8b library [33] (named

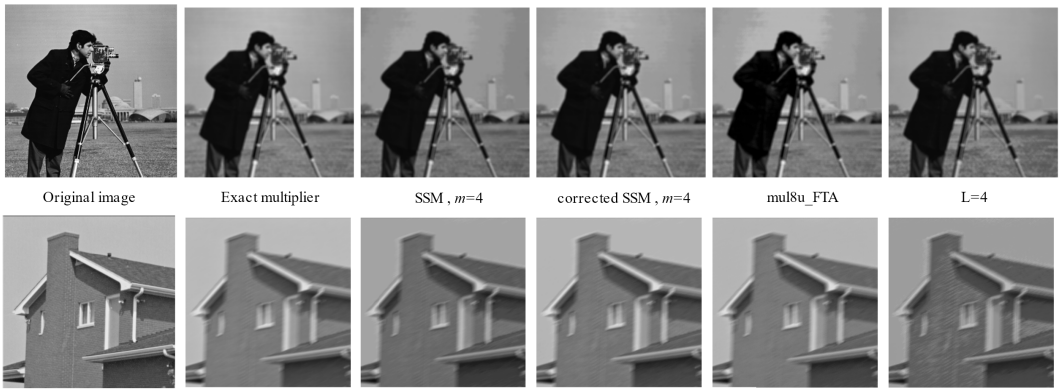

Fig. 11. Image filtering with unsigned multipliers. Top row: gaussian smoothing of "cameraman" image. Bottom row: processing of the "house" image with motion filter. Left: original image; second left: reference result, obtained by using an exact multiplier; other images: results using approximate multipliers.

mul8s_1L1G, *mul8s_1L2D*, *mul8s_1KVL*) and the two signed multipliers of [25] (Proposed and Hybrid), in the CN version (the signed FL versions using approximate compressors in all the partial-product matrix was not considered due to the poor error performance shown in [25]). The other multipliers are the same investigated in the previous subsection ([9], [12], [13], [28], [24], [26]) using sign-modulus conversion.

The results in Fig. 10 show some differences compared to the unsigned multipliers of Fig. 9. In multipliers using approximate sign-modulus conversion, the effect of the additional error introduced by the conversion is evident in the *m8_5*, *m8_6*, *Pro4* and *Pro5* multipliers (that where among the most accurate unsigned multipliers).

From Fig. 10(a) and Fig. 10(b), the SSMs using the proposed segmentation and the simplified error correction and $m = 4{,}5$ are on the Pareto frontier in both NMED-power and MRED-power both tradeoff plots. The SSMs using sign-modulus conversions turn out to be less effective as they are below the optimal curve.

Among the selected EvoApprox8b multipliers, *mul8s_1L1G*, *mul8s_1L2D* are on the Pareto curve in terms of NMED while they are below the frontier considering the MRED. Only the *m8_4* sign-modulus multipliers of [12] is close to the optimal curve, while the multipliers in [13] and the PM1 [28] and Mul_1 [24] perform well. The multipliers of [25], using approximate 4-2 compressors and without sign-modulus conversion, show the best NMED values, however these multipliers turn out to be less effective in terms of MRED (different from the unsigned counterparts).

It can be concluded that sign-modulus conversion is not convenient when applied to signed multipliers having reduced errors (such as *m8_5*, *m8_6*, *Pro4*, *Pro5*) while is an acceptable approach when using low-precision / low-power structures. The signed SSMs with the proposed segmentation and $m = 4{,}5$ are, again, in good position in terms of both NMED and MRED. These approximate multipliers, moreover, show a very good power reduction and hence emerge as a good choice in applications where their error performance is acceptable.

VI. Application Examples

We report some examples of image processing and recognition using investigated approximate multipliers, with either unsigned or signed versions.

A. Image Filtering With Unsigned Multipliers

In this application the output image Y is obtained by performing a convolution between the original image I and a suitable kernel, h:

$$Y(i,j)=\frac{1}{N}\sum_{m=-d}^{d}\sum_{n=-d}^{d} I(i+m,j+n)h(m+d+1,n+d+1) \tag{21}$$

The kernel h is a matrix of dimensions $(2d+1)\times(2d+1)$ whose coefficients are generally normalized so that:

$$\sum_{m=1}^{2d+1}\sum_{n=1}^{2d+1} h(m,n) = N \tag{22}$$

In our experiment we use two different kernels. The first one, h_1, performs image smoothing with a symmetric Gaussian lowpass filter of size 5 with standard deviation 2. It is obtained starting with the matlab command [50]: *fspecial*('*gaussian*',5,2) that produces a floating-point kernel with values normalized to 1. These values are multiplied by $N = 2^{11} = 2048$ and then rounded to integer values, thus setting the kernel in an appropriate range for our 8×8 multipliers. The second kernel, h_2, is a filter that approximates the effect of the linear motion of a camera by 10 pixels, with an angle of 25 degrees. The h_2 kernel was obtained in matlab with: *fspecial*('*motion*',10,25), with results multiplied by $N = 2^{11}$ and then rounded to integer values. The two kernels h_1 and h_2 are shown in Fig. 12. They include only positive values and therefore the convolution in (21) can be carried-out with unsigned multipliers (note that division by N is obtained in hardware by right-shifting the convolution result). In our experiment, we considered nine 8-bit grayscale images.

The Fig. 11 shows some examples of image filtering; in addition to the results obtained with an exact multiplier (reference image) the images produced by the four less power-consuming unsigned approximate multipliers are shown. As it can be observed, the results are very close to the one produced by the exact multiplier. This is quantitatively confirmed by the results reported in Table VI. In this table the peak signal-to-noise ratio (PSNR) and the mean structural similarity index (SSIM) are used as metrics to compare the quality of the images obtained by using approximate multipliers with the

TABLE VI
IMAGE FILTERING RESULTS

	Motion filter		Gaussian smoothing		Average	
MULTIPLIER	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR
SSM, m=4	0.974	24.5	0.979	25.0	0.976	24.7
C-SSM, m=4	0.990	36.9	0.992	31.9	0.991	34.4
SSM, m=5	0.994	33.8	0.993	29.3	0.994	31.5
C-SSM, m=5	0.996	43.3	0.996	42.5	0.996	42.9
SSM, m=6	0.998	42.1	0.998	38.4	0.998	40.3
C-SSM, m=6	0.998	51.3	0.998	50.0	0.998	50.7
Kulkarni [9]	0.995	40.5	0.997	42.9	0.996	41.7
mul8u_185Q [33]	0.998	51.6	0.998	51.2	0.998	51.4
mul8u_ZFB [33]	0.999	56.3	0.999	57.0	0.999	56.6
mul8u_12N4 [33]	0.996	43.2	0.995	43.5	0.995	43.3
mul8u_FTA [33]	0.982	35.0	0.973	34.1	0.977	34.5
m8_1 [12]	0.964	34.3	0.977	33.5	0.971	33.9
m8_2 [12]	0.959	27.8	0.965	28.3	0.962	28.1
m8_3 [12]	0.998	50.3	0.998	50.9	0.998	50.6
m8_4 [12]	0.997	42.9	0.997	46.1	0.997	44.5
m8_5 [12]	1.000	64.5	1.000	65.0	1.000	64.7
m8_6 [12]	0.999	59.4	1.000	62.7	1.000	61.1
L=2 [13]	0.996	43.2	0.997	44.4	0.997	43.8
L=3 [13]	0.989	34.3	0.995	38.9	0.992	36.6
L=4 [13]	0.952	25.6	0.964	27.6	0.958	26.6
PM1 [28]	0.994	43.9	0.998	50.2	0.996	47.1
Mul_1 [24]	0.972	33.2	0.987	36.8	0.980	35.0
Pro4 [26]	0.999	58.4	0.999	59.2	0.999	58.8
Pro5 [26]	1.000	65.7	1.000	73.0	1.000	69.3
Prop-FL [25]	0.999	53.6	0.999	57.8	0.999	55.7
Hybrid-FL [25]	0.998	52.5	0.998	51.8	0.998	52.1
Prop-CN [25]	1.000	60.9	1.000	61.9	1.000	61.4
Hybrid-CN [25]	0.999	60.1	0.999	58.2	0.999	59.1

$$h_1 = \begin{pmatrix} 48 & 69 & 78 & 69 & 48 \\ 69 & 101 & 114 & 101 & 69 \\ 78 & 114 & 129 & 114 & 78 \\ 69 & 101 & 114 & 101 & 69 \\ 48 & 69 & 78 & 69 & 48 \end{pmatrix} \quad h_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 & 91 & 177 & 14 \\ 0 & 0 & 0 & 0 & 0 & 18 & 104 & 189 & 128 & 43 & 0 \\ 0 & 0 & 0 & 31 & 116 & 201 & 116 & 31 & 0 & 0 & 0 \\ 0 & 43 & 128 & 189 & 104 & 18 & 0 & 0 & 0 & 0 & 0 \\ 14 & 177 & 91 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Fig. 12. Kernels for gaussian smoothing (left) and for motion effect (right).

reference one. These values reflect the degradation due to utilization of approximate multipliers. The table reports, for gaussian smoothing and motion filter, the average PSNR and SSIM values obtained with the nine test images. The last two columns in Table VI report values averaged also between the two filters.

As it can be observed, all the investigated approximate multipliers work well in these applications. The lowest SSIM is produced by L=4 multiplier, and it is very acceptable, about 0.96. The lowest PSNR, given by SSM with m=4, is about 24dB. Best results are given by Pro5 multiplier, with SSIM close to 1 and PSNR larger than 69dB. The SSM with m = 4 and the proposed correction circuit yields an SSIM of 0.99 with a PSNR larger than 30dB, with 70% power saving compared to the exact multiplier, and hence can be considered as the best solution for this application.

B. Edge Detection Using Sobel Operator

Edge detection finds several applications in computer vision for low-level feature extraction [51]. In this application the x and y component of the gradient of the image I (indicated as G_X and G_Y) are computed by convolving the image with two kernels S_X and S_Y. The edges of the image are extracted from the gradient magnitude, given by:

$$G = \sqrt{G_X^2 + G_Y^2} \tag{23}$$

The kernel S_X (corresponding to a 5×5 Sobel template [52]), is given by:

$$S_X = \begin{pmatrix} 1 & 2 & 0 & -2 & -1 \\ 4 & 8 & 0 & -8 & -4 \\ 6 & 12 & 0 & -12 & -6 \\ 4 & 8 & 0 & -8 & -4 \\ 1 & 2 & 0 & -2 & -1 \end{pmatrix} \tag{24}$$

while S_Y is the transpose of S_X. As in [25], image convolutions with kernels S_X and S_Y is performed with approximate signed multipliers, while square and square root in (23) are exact. The Fig. 13 shows some results obtained with the "cameraman" image, while averaged PSNR and SSIM obtained for the nine test images are shown in Table VII.

TABLE VII
EDGE DETECTION RESULTS

	Average	
MULTIPLIER	SSIM	PSNR
SSM, m=4	0.16	10.7
C-SSM, m=4	0.38	13.5
SSM, m=5	0.95	38.7
C-SSM, m=5	0.93	36.9
SSM, m=6	0.99	45.1
C-SSM, m=6	0.98	43.7
sign/modul. C-SSM m=4	0.66	20.3
sign/modul. C-SSM m=5	0.66	20.0
sign/modul. C-SSM m=6	0.66	19.8
sign/modul.Kulkarni [9]	0.64	19.5
mul8s_1KVL [33]	0.74	28.5
mul8s_1L2D [33]	0.36	18.3
mul8s_1L1G [33]	0.08	4.9
sign/modul. m8_1 [12]	0.62	19.6
sign/modul. m8_2 [12]	0.60	19.0
sign/modul. m8_3 [12]	0.62	19.6
sign/modul. m8_4 [12]	0.60	19.0
sign/modul. m8_5 [12]	0.66	19.7
sign/modul. m8_6 [12]	0.63	19.2
sign/modul. L=2 [13]	0.63	19.3
sign/modul. L=3 [13]	0.52	16.7
sign/modul. L=4 [13]	0.50	16.7
sign/modul. PM1 [28]	0.63	19.2
sign/modul. Mul_1 [24]	0.52	17.6
sign/modul. Pro4 [26]	0.61	18.8
sign/modul. Pro5 [26]	0.66	19.7
Prop-CN [25]	0.74	28.2
Hybrid-CN [25]	0.72	28.1

In this application the SSMs with $m = 4$ does not give acceptable results; the $m = 4$ SSM with correction performs slightly better but with results far from the exact multiplier. On the other hand, the SSMs with $m = 5$ yields an output very close to the exact one. All the signed multipliers using approximate sign/modulus transformation give similar result, with SSIM in the range 0.6-0.7 and PSNR around 20dB, except for the multipliers L = 3 and L = 4 of [13], with SSIM about 0.5 and PSNR about 16dB. The mul8s_1KVL [33] works well in this application, while the other two EvoApprox8b multipliers do not provide good results. The multipliers of [25] show an SSIM of about 0.7 with about 28dB PSNR.

Fig. 13. Edge detection using the Sobel operator, Top left: original "cameraman" image. Top, second left: reference result, obtained by using an exact multiplier. Other images: results obtained with approximate multipliers.

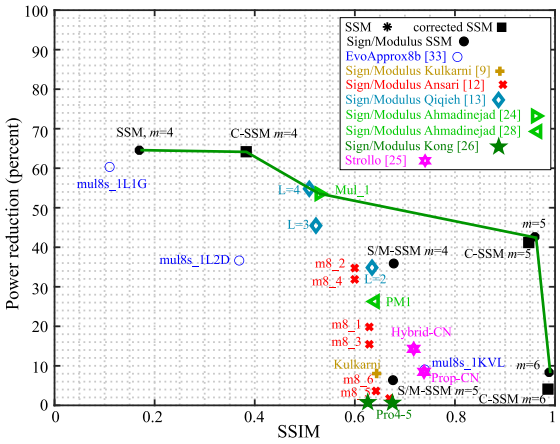

Fig. 14. Tradeoff between power saving and SSIM for edge detection application. Each point corresponds to a multiplier, the closer the point to the top-right corner, the better the tradeoff. The green line is the Pareto-optimal frontier, given by multipliers with the best power-SSIM tradeoff.

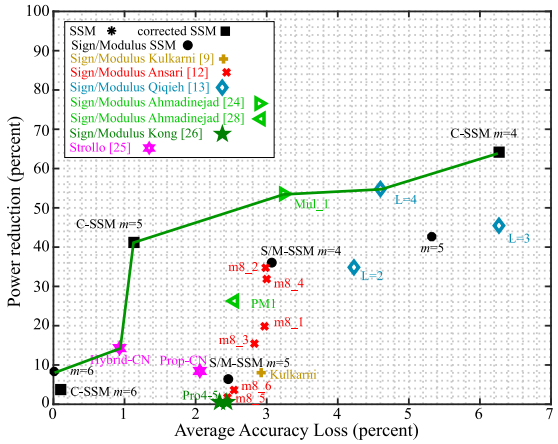

Fig. 15. Tradeoff between power saving and image recognition performance. Each point corresponds to a multiplier, the closer the point to the top-left corner, the better the tradeoff. The green line is the Pareto-optimal frontier, given by multipliers with the best power-accuracy tradeoff.

The Fig. 14 shows the tradeoff between power saving and SSIM for the edge detection application. The SSMs with $m = 5$ give very good results, with SSIM close to 0.95 and power saving larger than 40%. A larger power saving, close to 55%, can be obtained by Mul_1 and L = 4 multipliers that, however, allow to obtain an SSIM only slightly higher than 0.5.

C. Image Recognition Using MLP Classifier

Neural networks are widely used in machine learning, a typical application being image classification. We performed some image classification experiments using multilayer perceptron (MLP) classifiers. Two datasets have been considered. The first one is the MNIST, a dataset of handwritten 28 × 28 pixels images of handwritten numbers, containing 60,000 images for training and 10,000 images for testing [53]. The second one is Fashion-MNIST, a dataset of Zalando's article images, with ten classes, that shares the same image size and structure of training and testing splits of MNIST [54].

The employed MLPs have 784 inputs (one for each pixel of the images), two hidden layers of 80 and 60 neurons, and 10 outputs representing the probability that the image belongs to a one of the classes, for a total of 68,120 weights. The weights are signed numbers. Two variants of the MLP have been implemented. The first one uses the hyperbolic tangent activation function, while the sigmoid is used in the second one. The two MLPs versions allow us to investigate the performance of approximate multipliers in this kind of application for different operands distributions.

The training of the MLPs has been performed in MATLAB, using floating point arithmetic. After training, the MLP weights are quantized on 8-bits and the classification accuracy for each of the investigated multipliers is computed. Signed

TABLE VIII
IMAGE RECOGNITION RESULTS

	MNIST-tanh		MNIST-sigmoid		Fashion-tanh		Fashion-sigmoid		**Average Loss**
MULTIPLIER	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	
Floating point	97.26%	–	97.32%	–	89.34%	–	87.64%	–	–
Exact 8-bit	97.28%	-0.02%	97.29%	0.03%	89.25%	0.09%	87.70%	-0.06%	0.01%
SSM, m=4	11.85%	85.41%	29.67%	67.65%	10.00%	79.34%	35.72%	51.92%	71.08%
C-SSM, m=4	90.37%	6.89%	90.58%	6.74%	81.81%	7.53%	83.72%	3.92%	6.27%
SSM, m=5	94.99%	2.27%	95.90%	1.42%	74.85%	14.49%	84.53%	3.11%	5.32%
C-SSM, m=5	96.62%	0.64%	96.85%	0.47%	87.47%	1.87%	86.09%	1.55%	1.13%
SSM, m=6	97.31%	-0.05%	97.15%	0.17%	89.49%	-0.15%	87.57%	0.07%	0.01%
C- SSM, m=6	97.20%	0.06%	97.21%	0.11%	89.25%	0.09%	87.49%	0.15%	0.10%
sign/modul.C-SSM m=4	95.78%	1.48%	96.95%	0.37%	79.95%	9.39%	86.60%	1.04%	3.07%
sign/modul. C-SSM m=5	96.66%	0.60%	97.06%	0.26%	81.48%	7.86%	86.48%	1.16%	2.47%
sign/modul. C-SSM m=6	96.79%	0.47%	97.07%	0.25%	81.60%	7.74%	86.46%	1.18%	2.41%
sign/modul.Kulkarni [9]	96.30%	0.96%	96.79%	0.53%	80.52%	8.82%	86.26%	1.38%	2.92%
mul8s_1KVL [33]	54.55%	42.71%	75.92%	21.40%	18.32%	71.02%	64.27%	23.37%	39.63%
mul8s_1L2D [33]	72.37%	24.89%	86.69%	10.63%	29.71%	59.63%	72.24%	15.40%	27.64%
mul8s_1L1G [33]	12.15%	85.11%	27.47%	69.85%	10.00%	79.34%	41.51%	46.13%	70.11%
sign/modul. m8_1 [12]	96.01%	1.25%	96.99%	0.33%	80.46%	8.88%	86.22%	1.42%	2.97%
sign/modul. m8_2 [12]	95.95%	1.31%	96.98%	0.34%	80.19%	9.15%	86.50%	1.14%	2.99%
sign/modul. m8_3 [12]	96.31%	0.95%	96.95%	0.37%	80.60%	8.74%	86.39%	1.25%	2.83%
sign/modul. m8_4 [12]	96.10%	1.16%	96.87%	0.45%	80.30%	9.04%	86.29%	1.35%	3.00%
sign/modul. m8_5 [12]	96.70%	0.56%	97.11%	0.21%	81.51%	7.83%	86.45%	1.19%	2.45%
sign/modul. m8_6 [12]	96.66%	0.60%	97.03%	0.29%	81.24%	8.10%	86.47%	1.17%	2.54%
sign/modul. L=2 [13]	95.35%	1.91%	96.67%	0.65%	76.57%	12.77%	86.06%	1.58%	4.23%
sign/modul. L=3 [13]	93.30%	3.96%	95.93%	1.39%	71.82%	17.52%	85.44%	2.20%	6.27%
sign/modul. L=4 [13]	95.16%	2.10%	96.54%	0.78%	75.57%	13.77%	85.89%	1.75%	4.60%
sign/modul. PM1 [28]	96.59%	0.67%	97.06%	0.26%	81.26%	8.08%	86.47%	1.17%	2.55%
sign/modul. Mul_1 [24]	95.23%	2.03%	96.60%	0.72%	80.98%	8.36%	85.79%	1.85%	3.24%
sign/modul. Pro4 [26]	96.77%	0.49%	97.10%	0.22%	81.74%	7.60%	86.51%	1.13%	2.36%
sign/modul. Pro5 [26]	96.78%	0.48%	97.10%	0.22%	81.55%	7.79%	86.45%	1.19%	2.42%
Prop-CN [25]	94.00%	3.26%	95.72%	1.60%	86.39%	2.95%	87.21%	0.43%	2.06%
Hybrid-CN [25]	95.53%	1.73%	96.41%	0.91%	88.34%	1.00%	87.56%	0.08%	0.93%

multipliers are employed in this step, as MLP weights are both negative and positive. Classification results are reported in Table VIII.

The columns "Loss" in this table report the accuracy loss, in percentage, compared to floating point-multiplier. The last column is the average loss for the four considered cases. The accuracy obtained by using floating point multiplier reported in Table VIII are congruent with typical MLP classification performance reported in [53],[54]. For the MNIST dataset the sigmoid activation function gives slightly better results compared to hyperbolic tangent, while for Fashion-MNIST the hyperbolic tangent activation function gives more than 1.5% improvement in accuracy detection.

Weights quantization does not result in accuracy degradation, in some cases some minor accuracy improvement is observed [55]. For SSMs, the error correction reveals effective in this application. For $m = 4$ error correction decreases the average accuracy loss from 71% to about 6%, while for $m = 5$ the accuracy loss is reduced from about 5% to 1%. For $m = 6$ the classification is very close to the one of exact multiplier, and error correction is ineffective (a slight worsening of the accuracy, less than 0.1%, is observed). The EvoApprox8b multipliers do not give good results in this application; the best multiplier is mul8s_1L2D with a classification accuracy ranging from 87% to a mere 30%. Multipliers using approximate sign-modulus conversion all give similar classification accuracy, with accuracy loss in the range 2-3%; the multipliers of [13] give a higher accuracy loss between 4-6%. The multipliers of [25] give good classification accuracy, the Hybrid architecture showing accuracy loss lower than 1%.

To put these results in perspective, the Fig. 15 shows the tradeoff between power saving and average accuracy loss. The corrected SSMs with $m = 5$ is on the Pareto frontier, with accuracy loss of about 1% and more than 40% power saving. The corrected SSMs with $m = 4$ is also on the optimal curve, giving about 64% power saving with an accuracy loss of about 6%. The $m = 6$ SSM performs as the exact multiplier, albeit with a power reduction limited to around 8%. Among the other approximate multipliers, Mul_1 and L=4 give good tradeoff between power saving (about 55%) and accuracy loss.

VII. CONCLUSION

A detailed analysis of Static Segmented Multipliers has been performed in the paper. We have investigated both signed and unsigned multipliers, and for the latter a new segmentation approach has been presented. We have also developed simple and effective error correction techniques, able to significantly reduce the approximation error, at reduced hardware cost. We have performed a detailed comparison with previously proposed approximate multipliers that shows that the SSMs with the proposed correction technique have the desirable characteristic of being on (or close to) the Pareto-optimal frontier for both power vs NMED and power vs MRED trade-off plots. These multipliers, therefore, are good candidates

in applications where their error performance is acceptable. This is confirmed by the results obtained for image processing and image classification applications. Possible further work includes the analysis of recursive segmented multipliers (in which the inner multiplier is itself segmented) and segmented multipliers using an approximate inner multiplier.

REFERENCES

[1] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *Proc. 52nd Annu. Design Autom. Conf.*, Jun. 2015, pp. 1–6.

[2] Q. Xu, M. Todd, and S. K. Nam, "Approximate computing: A survey," *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, Feb. 2016.

[3] W. Liu, F. Lombardi, and M. Schulte, "A retrospective and prospective view of approximate computing," *Proc. IEEE*, vol. 108, no. 3, pp. 394–399, Mar. 2020.

[4] 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.

[5] 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.

[6] M. S. Ansari, B. F. Cockburn, and J. Han, "An improved logarithmic multiplier for energy-efficient neural computing," *IEEE Trans. Comput.*, vol. 70, no. 4, pp. 614–625, Apr. 2021.

[7] R. Pilipovic, P. Bulic, and U. Lotric, "A two-stage operand trimming approximate logarithmic multiplier," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 6, pp. 2535–2545, Jun. 2021.

[8] 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.

[9] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *Proc. 24th Int. Conf. VLSI Design*, Jan. 2011, pp. 346–351.

[10] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, and J. Henkel, "Architectural-space exploration of approximate multipliers," in *Proc. 35th Int. Conf. Comput.-Aided Design*, Nov. 2016, pp. 1–8.

[11] 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.

[12] 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.

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

[14] A. Cilardo *et al.*, "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.

[15] 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. (APCCAS)*, Oct. 2018, pp. 291–294.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Dual-quality 4:2 compressors for utilizing in dynamic accuracy configurable multipliers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 4, pp. 1352–1361, Apr. 2017.

[23] 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.

[24] 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, pp. 1–11, Oct. 2019.

[25] 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.

[26] T. Kong and S. Li, "Design and analysis of approximate 4–2 compressors for high-accuracy multipliers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 10, pp. 1771–1781, Oct. 2021.

[27] W. Guo and S. Li, "Fast binary counters and compressors generated by sorting network," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 6, pp. 1220–1230, Jun. 2021.

[28] M. Ahmadinejad and M. H. Moaiyeri, "Energy- and quality-efficient approximate multipliers for neural network and image processing applications," *IEEE Trans. Emerg. Topics Comput.*, early access, Apr. 13, 2021, doi: 10.1109/TETC.2021.3072666.

[29] N. Petra, D. De Caro, V. Garofalo, E. Napoli, and A. G. M. Strollo, "Design of fixed-width multipliers with linear compensation function," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 58, no. 5, pp. 947–960, May 2011.

[30] D. De Caro, N. Petra, A. G. M. Strollo, F. Tessitore, and E. Napoli, "Fixed-width multipliers and multipliers-accumulators with min-max approximation error," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 60, no. 9, pp. 2375–2388, Sep. 2013.

[31] G. Zervakis, K. Tsoumanis, S. Xydis, D. Soudris, and K. Pekmestzi, "Design-efficient approximate multiplication circuits through partial product perforation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 10, pp. 3105–3117, Oct. 2016.

[32] 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.

[33] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 258–261.

[34] V. Mrazek, Z. Vasicek, L. Sekanina, H. Jiang, and J. Han, "Scalable construction of approximate multipliers with formally guaranteed worst case error," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 11, pp. 2572–2576, Nov. 2018.

[35] 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.

[36] 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.

[37] M. H. S. Javadi, M. H. Yalame, and H. R. Mahdiani, "Small constant mean-error imprecise adder/multiplier for efficient VLSI implementation of MAC-based applications," *IEEE Trans. Comput.*, vol. 69, no. 9, pp. 1376–1387, Sep. 2020.

[38] T. Yang, T. Sato, and T. Ukezono, "An approximate multiply-accumulate unit with low power and reduced area," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2019, pp. 385–390.

[39] D. Esposito, A. G. M. Strollo, and M. Alioto, "Low-power approximate MAC unit," in *Proc. 13th Conf. Ph.D. Res. Microelectron. Electron. (PRIME)*, Jun. 2017, pp. 81–84.

[40] G. A. Gillani, M. A. Hanif, M. Krone, S. H. Gerez, M. Shafique, and A. B. J. Kokkeler, "SquASH: Approximate square-accumulate with self-healing," *IEEE Access*, vol. 6, pp. 49112–49128, 2018.

[41] H. Saadat, H. Bokhari, and S. Parameswaran, "Minimally biased multipliers for approximate integer and floating-point multiplication," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2623–2635, Nov. 2018.
[42] 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)*, Austin, TX, USA, Nov. 2015, pp. 418–425.
[43] 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.
[44] S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram, "LETAM: A low energy truncation-based approximate multiplier," *Comput. Elect. Eng.*, vol. 63, pp. 1–17, Oct. 2017.
[45] 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.
[46] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim, "Energy-efficient approximate multiplication for digital signal processing and classification applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 6, pp. 1180–1184, Jun. 2015.
[47] L. Li, I. Hammad, and K. El-Sankary, "Dual segmentation approximate multiplier," *Electron. Lett.*, vol. 57, no. 19, pp. 718–720, Sep. 2021.
[48] R. Jothin and C. Vasanthanayaki, "High performance modified static segment approximate multiplier based on significance probability," *J. Electron. Test.*, vol. 34, no. 5, pp. 607–614, Oct. 2018.
[49] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy, "Design of power-efficient approximate multipliers for approximate artificial neural networks," in *Proc. 35th Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2016, pp. 1–7.
[50] The MathWorks Inc. *MATLAB Version 9.11.0 (R2021b)*. Accessed: Oct. 27, 2021. [Online]. Available: https://www.mathworks.com/help/images/ref/fspecial.html
[51] M. S. Nixon and A. S. Aguado, *Feature Extraction and Image Processing for Computer Vision*, 3rd ed. Amsterdam, The Netherlands: Elsevier, 2012.
[52] *Developer Reference for Intel Integrated Performance Primitives-FilterSobel*. Accessed: Oct. 27, 2021. [Online]. Available: https://www.intel.com/content/www/us/en/develop/documentation/ipp-dev-reference/top/volume-2-image-processing/filtering-functions-2/fixed-filters/filtersobel.html
[53] Y. LeCun, C. Cortes, and C. Burges. *The MNIST Database of Handwritten Digit*. Accessed: Oct. 27, 2021. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[54] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.
[55] 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 Integr. (VLSI) Syst.*, vol. 28, no. 2, pp. 317–328, Oct. 2020.

Antonio Giuseppe Maria Strollo (Senior Member, IEEE) received the M.S. *(cum laude)* and Ph.D. degrees in electronic engineering from the University of Napoli Federico II, Italy. Since 2002, he has been a Full Professor with the University of Napoli Federico II, where he was the Head of the Department of Electronic and Telecommunication Engineering from 2005 to 2008. He has published more than 150 articles on international journals and conferences. His current research interests include arithmetic circuits, approximate computing, and low-power digital signal processing circuits. He has been a Technical Program Committee Member for international conferences, including PRIME, ICECS, and ESSCIRC/ESSDERC. He and his coauthors were a recipient of the 2021 IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS Guillemin-Cauer Best Paper Award. From 2009 to 2012, he served as an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS. He is currently an Associate Editor of *Integration, the VLSI Journal*.

Ettore Napoli (Senior Member, IEEE) received the degree (Hons.) in electronic engineering in 1995, the Ph.D. degree in electronic engineering in 1999, and the degree (Hons.) in physics in 2009.

He was a Research Associate with the Engineering Department, University of Cambridge, U.K., in 2004. He has been a Full Professor with the University of Napoli Federico II since 2020. He has authored or coauthored more than 100 articles published in international journals and conferences. His research interests include modeling and design of power semiconductor devices and VLSI circuit design.

Davide De Caro (Senior Member, IEEE) received the M.S. degree (Hons.) in electronic engineering and the Ph.D. degree in electronic engineering and computer science from the University of Naples Federico II, Italy, in July 1999 and February 2003, respectively.

He has worked in the area of digital integrated VLSI circuit design for the last 14 years. He is currently an Associate Professor at the Department of Electrical Engineering and Information Technology, University of Naples Federico II. He is the author of more than 80 technical papers in international journals and refereed international conferences.

Nicola Petra (Member, IEEE) received the M.S. and Ph.D. degrees from the University of Napoli Federico II in 2002 and 2007, respectively. He is currently working as an Associate Professor at the Department of Electrical Engineering and Information Technology, University of Napoli Federico II. He has authored or coauthored more than 60 papers on scientific journals and international conferences. His research interests include design of digital VLSI circuits for telecommunications and high-performance arithmetic circuits.

Gerardo Saggese received the M.Sc. degree 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 pursuing the Ph.D. 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 circuit.

Gennaro Di Meo received the M.S. degree *(cum laude)* from the University of Napoli Federico II, Italy, in 2018, where he is currently pursuing the Ph.D. degree in electrical engineering and information technology. His research interests include design of digital VLSI circuits for telecommunications, LMS filters, and approximate computing.