Novel Low-Power Floating-Point Divider With Linear Approximation and Minimum Mean Relative Error

Gennaro Di Meo, Antonio Giuseppe Maria Strollo, *Senior Member, IEEE*, and Davide De Caro, *Senior Member, IEEE*

***Abstract*— Floating-point division involves the computation of the ratio (1 + *Mx*)/(1 + *My*), where *Mx* and *My* represents the mantissas of the input values. In this paper, we propose a new method for approximating this operation using a linear function of *Mx*, with coefficients that depend on *My*. The coefficients are calculated to minimize the Mean Relative Error Distance (*MRED*) of the approximation. To this end, the range of My is partitioned in N sub-intervals where the minimization of *MRED* is formulated as a linear programming problem, whose solution gives optimal coefficient values. The hardware implementation requires a small lookup table, two multipliers and an adder. An aggressive coefficients quantization is exploited to further optimize the design. Obtained *MRED* improves by increasing *N*, ranging from 1.4% to 0.33%. Implementation results in a 28nm CMOS technology show that the proposed design outperforms the state-of-the-art, offering the best trade-off between hardware complexity and accuracy. Results for two image processing applications, change detection and JPEG compression, demonstrate remarkable performance, with SSIM very close to 1 and PSNR values exceeding 50dB.**

***Index Terms*— Floating-point divider, approximate computing, low-power technique, error minimization.**

I. Introduction

ARITMETIC circuits play a key role in the design of digital signal processing (DSP) algorithms, ubiquitous in daily electronic applications. The arise of artificial intelligence and big data processing, which demands for operations as recognition, classification, or machine learning, calls for an intensive usage of arithmetic operations [1]. Recent systems based on Internet of Things (IoT) paradigm also need to process, store, and transmit massive amount of data, making the design of electronic devices with low-power features challenging [2], [3].

Since adders, multipliers, and dividers are energy-consuming circuits, the adoption of suitable design strategies has become a priority in order to realize target tasks with acceptable power consumption.

Manuscript received 19 May 2023; revised 25 July 2023; accepted 30 August 2023. Date of publication 5 October 2023; date of current version 18 December 2023. This article was recommended by Associate Editor X. S. Zhang. *(Corresponding author: Gennaro Di Meo.)*

The authors are with the Department of Electrical Engineering and Information Technology, University of Naples Federico II, 80125 Naples, Italy (e-mail: gennaro.dimeo@unina.it).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2023.3312974.

Digital Object Identifier 10.1109/TCSI.2023.3312974

In this scenario, Approximate Computing (AC) constitutes a valuable solution allowing to reduce area and power at the cost of accepting errors in the computation [4], [5]. In addition, the limit of human senses and the error-tolerant nature of many practical applications (as image and audio processing, or adaptive filtering) make the AC approach very effective [6], [7], [8].

Several works have been dedicated to the design of fixed-point approximate adders and multipliers, proposing a plethora of techniques able to optimize power and area. For instance, the papers [9], [10], [11] show a decomposition method that divides the adder in atomic fast sub-adders, each one working on a portion of the input signals, while [12], [13], [14] exploit an approximate carry-skip architecture able to reduce the critical path delay. In [15] the speculation method is applied to parallel-prefix adders, while [16], [17] present approximate full-adders both at gate and transistor level.

In case of multipliers, reducing the complexity of the partial product matrix (PPM) compression stage generally leads to remarkable power benefits. Again, several techniques have been proposed, ranging from approximate compression [18], [19], [20], [21], [22] to truncation [23], [24] or input segmentation [25], [26], [27], [28], [29], and suitable correction techniques are also described for accuracy recovery (see [20], [23], [26] for reference).

Unlike adders and multipliers, dividers have received less attention in literature. However, in the design of several commercial microprocessors and products [30], [31], [32], hardware dividers are preferred to software realization of the division.

The division between two fixed-point numbers generally exploits iterative algorithms based on subtractions/multiplications in order to compute the quotient starting from an initial estimate [33], [34], [35], [36], [37], [38]. In this case, latency and power consumption are primary concerns in the design. Algorithms as the Sweeney-Robertson-Tocher (SRT) try to reduce the number of iterations involving high-radix coding and redundant representations of the quotient [38]. Further approaches achieve power improvements approximating the subtractor [39], [40] or applying signal segmentation [41]. The realization of non-iterative dividers constitutes a further solution able to compute the quotient with low energy and reduced latency. In this case, the logarithmic number system (LNS) is a valuable means since it allows to express the division as two-operand subtraction followed by a shift [42]. In [43], the divisor y is recoded in order to employ only a multiplication and a left-shift, while [44] exploits a linear approximation for the term $1/y$. LNS with mean-error compensation is proposed in [45], whereas [46] devises a rounding-based approach to simplify the divider.

Floating-point arithmetic, which represents numbers with sign, exponent, and mantissa, offers both large dynamic range and fine accuracy [47]. These properties make floating-point divider design important for many practical DSP applications.

In a hardware divider, sign and exponent computation are simple to implement, involving only a XOR and a subtraction. On the other hand, the mantissa computation is much more complex, requiring a fixed-point division: $(1 + Mx)/(1 + My)$, where Mx and My are the mantissas of dividend and divisor, respectively. A two-step approximate technique is proposed in [48] to perform the mantissa division by means of shift-and-add operations. In this case, the amount of shift and the number of additions, defined at design time, allow to tune the tradeoff between precision and hardware complexity. In [49] a piecewise constant approximation is exploited. Like [48], different levels of accuracy can be achieved by properly choosing the number of ranges in which the constant approximation is applied. In [50] the mantissa division is approximated by means of subtractions and a variable correction term, stored in a LUT, is employed to recover precision. In this case, the number of bits of the correction term is a critical design parameter, since it impacts both the accuracy and the LUT size. In [51] the division is revisited as a two-variable function and best-fitting planes are used to approximate the surface of the quotient.

In this paper, we propose a novel approximate floating-point divider (named FPDME in the following), that is non-iterative and has minimal error. In our approach we start by considering the exact operation $(1 + Mx)/(1 + My)$, and we express the division as a linear function of the mantissa Mx, with coefficients depending on My.

The choice of coefficients affects the accuracy of the divider. In our approach, the coefficients are determined in order to minimize the Mean Relative Error Distance (*MRED*) of the approximation. To this end, the range of My is partitioned in N sub-intervals and in each sub-interval the minimization of *MRED* is formulated as a linear programming problem, whose solution gives optimal coefficient values. While we considered *MRED* minimization, it is worth noting that our proposed approach can be easily modified to target error metrics, such as mean absolute error, for example.

Mantissa truncation and coefficient quantization are also exploited to further optimize the design.

From a hardware perspective, the proposed divider requires only a lookup table (LUT), used to store the coefficients, and two multipliers and an adder, fused in a unique carry-save arithmetic structure. Suitable choice of N and of parameter quantization allow to tune at design-time the tradeoff between hardware complexity and accuracy.

The proposed FPDME allows to achieve *MRED* comparable or better than previously proposed approximate floating-point dividers. Synthesis results in TSMC 28nm CMOS technology also highlight an improvement of hardware performances with respect to the state-of-the-art, measured in terms of power-delay product (PDP) and area-delay product (ADP). We present results for two image processing applications: change detection and JPEG compression. Both applications further remark the advantages of the proposed technique, exhibiting competitive performances in terms of peak signal-to-noise ratio (PSNR) and Mean Structural Similarity Index (SSIM).

The paper is organized as follows. Section II introduces the floating-point notation and main steps used to perform the division. Section III describes our approach for approximating the division, while the Section IV shows the hardware implementation. Afterwards, the results are discussed in Section V in terms of error metric and hardware assessment, whereas Section VI presents the achieved performances in change detection and JPEG compression applications. Finally, Section VII concludes the paper.

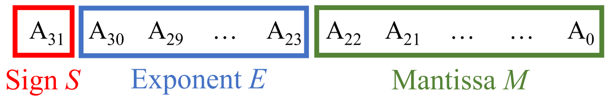


Fig. 1. Floating-point single-precision representation of the real number A.

II. Floating-Point Division

In floating-point notation, a real number A is represented as follows:

$$A = (-1)^S \cdot 2^{E-bias} \cdot (1 + M) \tag{1}$$

where S, E, and M are sign, exponent, and mantissa of A, respectively, whereas *bias* is a constant term used to shift the exponent. While one bit is used for the sign, the bit-width of E and M and the value of *bias* change in accordance with the desired precision. The Fig.1 shows the single precision IEEE-754 format [47]. The representation of A requires 32 bits, with E and M that are unsigned numbers expressed on 8 and 23 bits (highlighted in blue and green, respectively). The exponent E lies in the range [0, 255], whereas the mantissa M varies in the range [0, 1). In addition, *bias* is set to 127 in order to shift the overall exponent of (1) in the range [−127, 128].

In the following we assume that divider inputs are single-precision floating-point numbers, but the proposed technique is general and can be applied equally well to other floating-point formats such as IEEE half-precision or BFloat16.

In order to show the floating-point division, let us consider the two operands:

$$\begin{aligned} X &= (-1)^{Sx} \cdot 2^{Ex-bias} \cdot (1 + Mx) \\ Y &= (-1)^{Sy} \cdot 2^{Ey-bias} \cdot (1 + My) \end{aligned} \tag{2}$$

where Sx, Ex, and Mx are sign, exponent, and mantissa of the dividend, X, while Sy, Ey, My are sign, exponent, and mantissa of the divisor Y.

The division $Z = X/Y$ has a similar representation:

$$Z = (-1)^{Sz} \cdot 2^{Ez-bias} \cdot (1 + Mz) \tag{3}$$

where the mantissa Mz is normalized, assuming values in [0, 1). It is also worth noting that the quantity $(1 + Mz)$ lies in the range [1, 2). The sign Sz of the division is simply the XOR of the sign bit of the operands, whereas the modulus of Z can be written as:

$$|Z| = 2^{Ez-bias} \cdot (1 + Mz) = 2^{Ex-Ey} \cdot \frac{1+Mx}{1+My} \quad (4)$$

Let us consider the term $(1 + Mx)/(1 + My)$. Its maximum value is obtained for My very close to zero and Mx very close to one, resulting (slightly) less than 2. The minimum value is obtained in the opposite case and is (slightly) larger than 0.5. Therefore, the following inequality holds:

$$0.5 < \frac{1+Mx}{1+My} < 2 \quad (5)$$

In addition, it is worth noting that the factor $(1 + Mx)/(1 + My)$ is larger than 1 when the condition $Mx > My$ is true. Then, starting from (4) and (5), the following two cases are considered for the computation of Ez and Mz:

$$\begin{cases} Ez - bias = Ex - Ey \\ (1+Mz) = \dfrac{1+Mx}{1+My} \end{cases} \quad if\ Mx \geq My \quad (6)$$

$$\begin{cases} Ez - bias = Ex - Ey - 1 \\ (1+Mz) = 2\dfrac{1+Mx}{1+My} \end{cases} \quad if\ Mx < My \quad (7)$$

Indeed, the quotient $(1 + Mx)/(1 + My)$ is naturally in the interval [1, 2) when $Mx \geq My$ (see (6)). Conversely, $(1 + Mx)/(1 + My)$ is in the range [0.5, 1) when $Mx < My$. Therefore, in order to have $(1 + Mz)$ in [1, 2), the normalization process imposes to double $(1 + Mx)/(1 + My)$ and to subtract a '1' from the exponent for compensation as shown in (7).

Anyway, in both cases the mantissa computation requires the division $(1 + Mx)/(1 + My)$.

III. Proposed Floating-Point Divider

In this section we describe the technique used to approximate the divider. Firstly, we express the division $(1 + Mx)/(1 + My)$ as a linear function of the mantissa Mx, with coefficients that depend on My. Next, we obtain the coefficient values that optimize the *MRED* by solving a minimization problem formulated as a linear constrained programming problem. In a subsequent step, we perform an aggressive quantization of the coefficients to further optimize the design. To that purpose, we reformulate the optimization problem as an integer linear programming problem.

A. Division Approximated as a Linear Function of Mx

In order to show the proposed technique, let us first define the exact ratio as $f(Mx, My) = (1 + Mx)/(1 + My)$ and the approximate one as $\phi(Mx, My)$. The relative error distance (*RED*) between $f(Mx, My)$ and $\phi(Mx, My)$ is

$$RED = \left| \frac{f(Mx, My) - \phi(Mx, My)}{f(Mx, My)} \right| \quad (8)$$

while the *MRED* is the average value of *RED*.

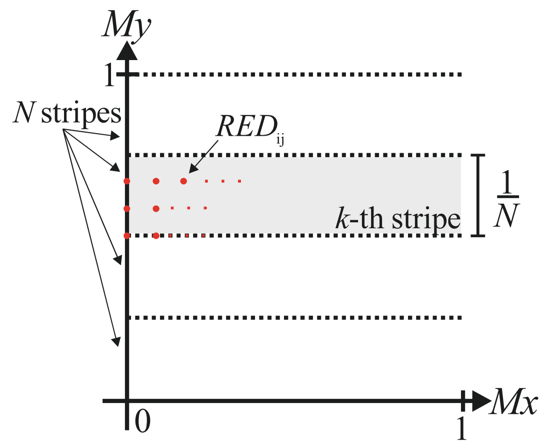


Fig. 2. Partition of the mantissas' plane in N stripes.

Let us also rewrite the division between mantissas as follows:

$$f(Mx, My) = \frac{1+Mx}{1+My} = \frac{1}{1+My} + \frac{1}{1+My} \cdot Mx \quad (9)$$

As shown in (9), $f(Mx, My)$ is linear with respect to Mx with coefficients that depend on My. Starting from this observation, we can write $f(Mx, My)$ as follows:

$$\phi(Mx, My) = g(My) + c(My) \cdot Mx \quad (10)$$

From (9)-(10) we should select $g(My) = c(My) = 1/(1 + My))$ to make the error equal to zero. However, $c(My)$ is to be multiplied by Mx to obtain the final result. Therefore, from the hardware implementation perspective, it makes sense to use two different approximations for $g(My)$ and $c(My)$, using a rougher approximation for $c(My)$.

With the above consideration in mind, we partition the range of My in N subintervals, each one having a width of $1/N$. This corresponds to divide the mantissas' plane $Mx - My$ in N horizontal stripes as shown in Fig.2. Note that we choose N as a power of two, so that each stripe can be easily identified by means of $h = \log_2(N)$ most significant bits (MSBs) of My.

In the k-th stripe $(k - 1)/N \leq My < k/N$ we approximate $c(My)$ with a constant: $c(My) = c_k$, while $g(My)$ is approximated with a linear function of My as follows: $g(My) = a_k + b_k\, My$.

Using the above assumptions, the equation (10) in the k-th stipe becomes:

$$\phi_k(Mx, My) = a_k + b_k \cdot My + c_k \cdot Mx \quad (11)$$

This equation requires a total of $3 \cdot N$ coefficients a_k, b_k and c_k to approximate the quotient and our goal becomes to compute the coefficients which minimize the *MRED*.

B. Obtaining the Optimal Coefficients

To obtain the values of the coefficients a_k, b_k and c_k, we discretize each stripe by considering $nx \times ny$ equally spaced points (highlighted in red in Fig. 2), in which the relative error distance is computed. Then, in a generic point

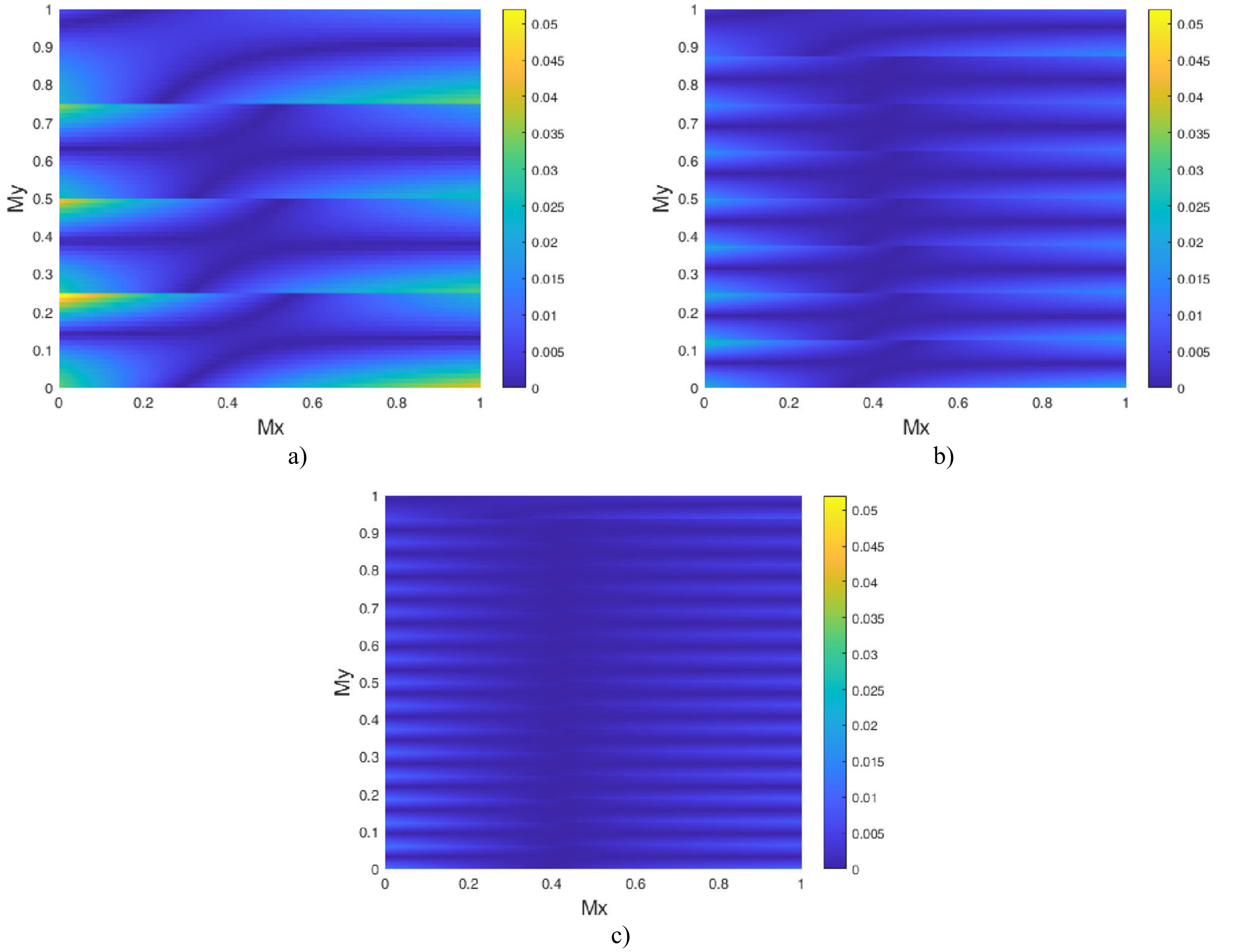


Fig. 3. 2D representation of *RED* in the mantissas' plane with (a) $N = 4$, (b) $N = 8$, and (c) $N = 16$.

of coordinates (Mx_i, My_j), the relative error distance $RED_{i,j}$ is expressed as:

$$RED_{i,j} = \left|\frac{f(Mx_i, My_j) - \phi_k(Mx_i, My_j)}{f(Mx_i, My_j)}\right| = \left|\frac{f(Mx_i, My_j) - a_k - b_k \cdot My_j - c_k \cdot Mx_i}{f(Mx_i, My_j)}\right| \tag{12}$$

with: $i = 0, 1, \ldots nx - 1$ and: $j = 0, 1, \ldots ny - 1$. Our problem can be formulated as follows: find the coefficients a_k, b_k, c_k in each stripe in order to minimize the following objective function:

$$\sum_{i=0}^{nx-1}\sum_{j=0}^{ny-1} RED_{i,j} \text{ min!} \tag{13}$$

It is worth noting that the summation in (13) corresponds to the *MRED* in the k-th stripe, except for a scaling factor. Therefore, minimizing (13) in each stripe allows to minimize the overall *MRED* of the divider. We also underline that other error metrics, not just *MRED*, could also be considered as a cost function in (12), (13), as an example the mean absolute error.

The optimization (13) can be further formulated as a linear programming problem by introducing some auxiliary variables u_{ij} such that:

$$\left|\frac{f(Mx_i, My_j) - a_k - b_k \cdot My_j - c_k \cdot Mx_i}{f(Mx_i, My_j)}\right| \leq u_{ij} \tag{14}$$

Then, posing $f_{ij} = f(Mx_i, My_j)$ for conciseness, (13) can be rewritten as:

$$\begin{aligned} &\sum_{i=0}^{nx-1}\sum_{j=0}^{ny-1} u_{ij} \text{ min!} \\ &subject\ to: \\ &-a_k - b_k \cdot My_j - c_k \cdot Mx_i - u_{ij} \cdot f_{ij} \leq -f_{ij} \\ &a_k + b_k \cdot My_j + c_k \cdot Mx_i - u_{ij} \cdot f_{ij} \leq f_{ij} \\ &for\ i = 0, 1, \ldots, nx - 1, \quad j = 0, 1, \ldots, ny - 1 \end{aligned} \tag{15}$$

where the constraints are derived from (14) after some algebra. The problem (15) takes the form of a standard linear programming problem of the form:

$$\begin{aligned} &\mathbf{c}^T\mathbf{x} \text{ min!} \\ &subject\ to: \mathbf{Ax} \leq \mathbf{b} \end{aligned} \tag{16}$$

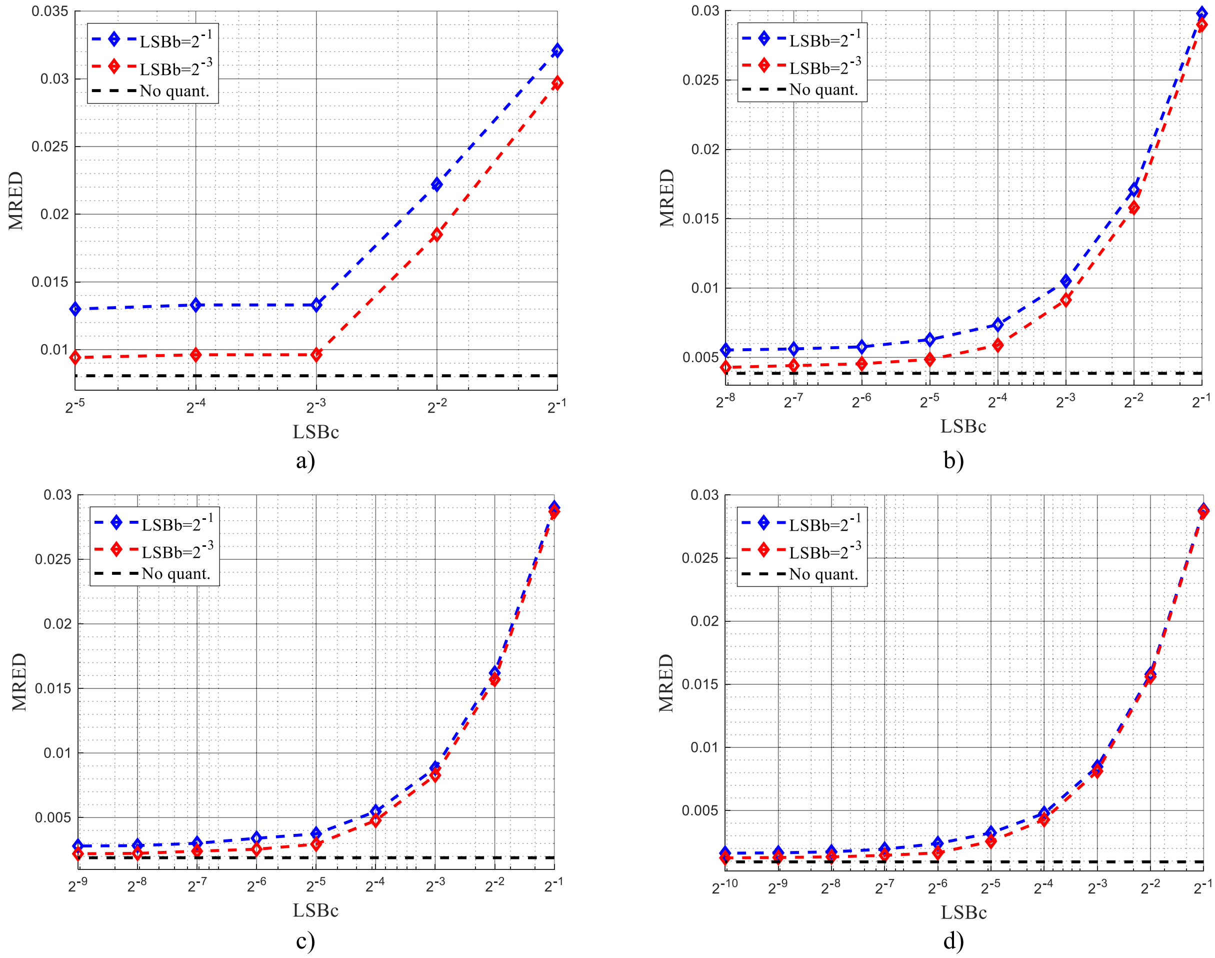


Fig. 4. *MRED* with respect to *LSBc* for $LSBa = 2^{-7}$ and $LSBb = 2^{-1}, 2^{-3}$ in the cases (a) $N = 4$, (b) $N = 8$, (c) $N = 16$, (d) $N = 32$.

where the unknown vector x is composed by $3 + nx \cdot ny$ elements (that are a_{k}, b_{k}, c_{k} and u_{ij} for $i = 0, 1, \ldots nx - 1$ and $j = 0, 1, \ldots ny - 1$) and the number of constraints is $2 \cdot nx \cdot ny$.

Figure 3a, 3b and 3c show the contour plot of *RED* for $N = 4$, 8 and 16, respectively, with the minimization problem solved in MATLAB using the *linprog* command. In the following, we assume $nx = 100$ and $ny = 20$. As shown, increasing N allows to achieve low values of *RED* in large regions of the mantissas' plane, as demonstrated by the blue sections that expand from $N = 4$ to $N = 16$. Accordingly, the *MRED* also improves by increasing the N value.

In addition, Fig. 3 suggests also to properly choose N in order to meet the desired accuracy constraints (dependent on the adopted floating-point format as an example).

C. Quantization of Coefficients

In order to realize the mantissa division in hardware, quantized values of the coefficients a_{k}, b_{k}, c_{k} are required. To that purpose, we rewrite a_{k}, b_{k}, c_{k} as follows:

$$\begin{aligned} a'_k &= a_{\mathrm{int},k} \cdot LSB_a \\ b'_k &= b_{\mathrm{int},k} \cdot LSB_b \\ c'_k &= c_{\mathrm{int},k} \cdot LSB_c \end{aligned} \tag{17}$$

where *LSBa*, *LSBb*, *LSBc* are the weights of the less-significant bits (LSB) of the coefficients (defined at design time), while $a_{\mathrm{int,k}}$, $b_{\mathrm{int,k}}$, $c_{\mathrm{int,k}}$ are integer variables, to be found.

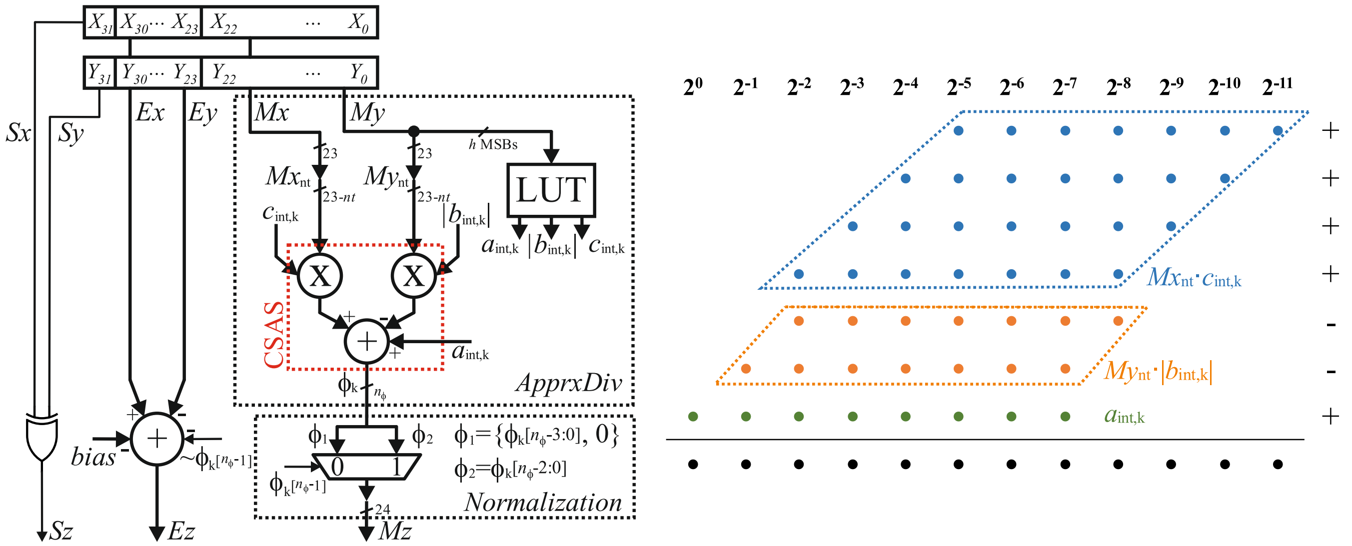
It is worth noting that the choice of *LSBa*, *LSBb*, *LSBc* can be properly tailored depending on the adopted floating-point format in order to meet the target accuracy.

By substituting a'_{k}, b'_{k}, c'_{k} to a_{k}, b_{k}, c_{k} in (15), we obtain a mixed-integer linear programming problem that can be solved in MATLAB with *intlinprog* command, giving the values of quantized coefficients that minimize the *MRED*.

Figure 4 shows the behavior of *MRED* when coefficients are quantized. In the figure, the *MRED* is function of *LSBc* for N varying between 4 and 32, with *LSBa* fixed to 2^{-7} and *LSBb* equal to 2^{-1} or 2^{-3}. We report also the error obtained with real (non-quantized) coefficients (see the black dashed line). In these simulations, the *MRED* is computed by considering 10^6 divisions, performed with 10^6 couples of uniform distributed numbers, expressed on 23 bits.

As shown in Fig. 4, the *MRED* exhibits a remarkable dependence on *LSBc* in all the cases. Indeed, a decrease in the values of *LSBc*, corresponding to finer resolutions of coefficients c'_k, leads to an improvement in precision, as expected.

On the other hand, a weaker dependence on *LSBb* is observed, particularly for $N \geq 16$, as shown in Fig. 4c and 4d.



a) b)

Fig. 5. a) Block diagram of the proposed FPDME and b) carry-save arithmetic structure with $N = 8$, $LSBa = 2^{-7}$, $LSBb = 2^{-1}$, $LSBc = 2^{-4}$, $nt = 16$.

TABLE I

COEFFICIENTS FOR N = 4, LSBA = 2^{-7}, LSBB = 2^{-1}, AND LSBC = 2^{-3}

h*=2 MSBs of *My	$a_{int,k}$	$b_{int,k}$	$c_{int,k}$
00	131	-2	7
01	139	-2	6
10	118	-1	5
11	128	-1	4

In this case, in fact, the *MRED* achieved for $LSBb = 2^{-3}$ is very close to the one achieved for $LSBb = 2^{-1}$.

In addition, a proper choice of *LSBa* also leads to satisfactory performances while being less critical for the design. In this case, we found that $LSBa = 2^{-7}$ is reasonable to achieve acceptable *MRED* for fine values of *LSBc*.

The results in Fig. 4 indicate that selecting *LSBc* as 2^{-3} for $N = 4$ and in the range 2^{-4}-2^{-7} for $N \geq 8$ results in acceptable error. Likewise, choosing $LSBb = 2^{-1}$ is also a reasonable option. Based on these observations, we focus our attention on the following test cases, with the aim to get both accurate results and moderate hardware complexity:

(i) $N = 4$, $LSBa = 2^{-7}$, $LSBb = 2^{-1}$, $LSBc = 2^{-3}$
(ii) $N = 8$, $LSBa = 2^{-7}$, $LSBb = 2^{-1}$, $LSBc = 2^{-4}$
(iii) $N = 16$, $LSBa = 2^{-7}$, $LSBb = 2^{-1}$, $LSBc = 2^{-4}$
(iv) $N = 32$, $LSBa = 2^{-7}$, $LSBb = 2^{-1}$, $LSBc = 2^{-5}$.

Tables I-IV collect the obtained values for the coefficients $a_{int,k}$, $b_{int,k}$, $c_{int,k}$, in the four considered cases.

IV. PROPOSED FLOATING-POINT DIVIDER

The hardware implementation of the proposed FPDME is depicted in Fig. 5a. The sign *Sz* is computed by XOR-ing *Sx* and *Sy*, whereas a multi-operand adder computes the exponent *Ez*. The approximate mantissa division is performed in the *ApprxDiv* block. The *h* MSBs of *My* index lookup table (LUT) that stores the quantized coefficients, while two multipliers and an adder compute the quotient. Since $b_{int,k}$ is always negative, we store in the LUT its absolute value $|b_{int,k}|$ in order to minimize LUT size. In any case, as shown by Tables I-IV, the LUTs are very small and do not require custom ROM. They have been described in Verilog HDL

TABLE II

COEFFICIENTS FOR N = 8, LSBA = 2^{-7}, LSBB = 2^{-1}, AND LSBC = 2^{-4}

h*=3 MSBs of *My	$a_{int,k}$	$b_{int,k}$	$c_{int,k}$
000	133	-3	15
001	130	-2	14
010	138	-2	12
011	117	-1	11
100	119	-1	10
101	118	-1	10
110	122	-1	9
111	128	-1	8

TABLE III

COEFFICIENTS FOR N = 16, LSBA = 2^{-7}, LSBB = 2^{-1}, AND LSBC = 2^{-4}

h*=4 MSBs of *My	$a_{int,k}$	$b_{int,k}$	$c_{int,k}$
0000	132	-3	15
0001	128	-2	15
0010	130	-2	14
0011	134	-2	13
0100	134	-2	13
0101	139	-2	12
0110	118	-1	11
0111	117	-1	11
1000	119	-1	10
1001	118	-1	10
1010	118	-1	10
1011	122	-1	9
1100	122	-1	9
1101	122	-1	9
1110	127	-1	8
1111	128	-1	8

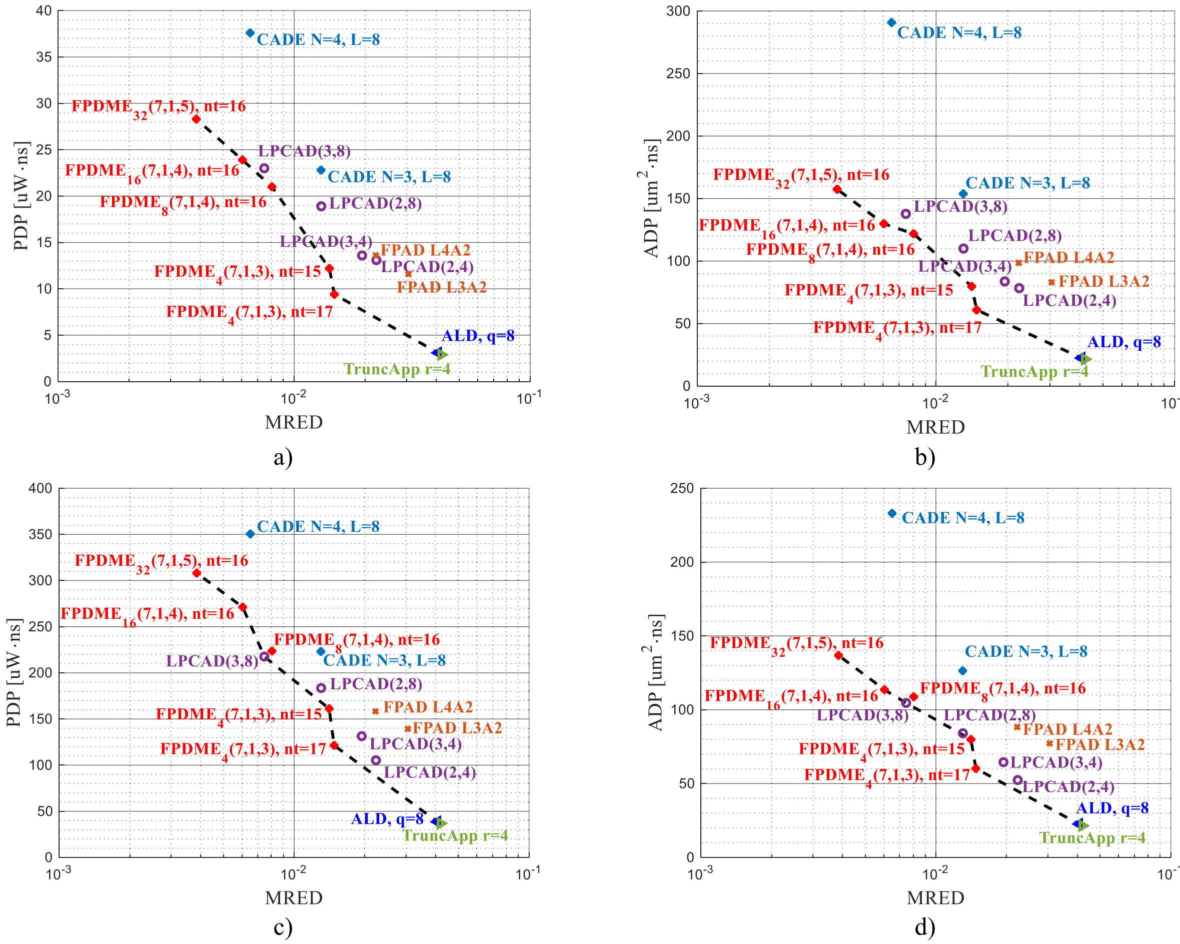


Fig. 6. a) PDP and b) ADP with respect to the *MRED* for minimum power and area implementations. c) PDP and d) ADP with respect to the *MRED* with a 750ps constraint on the maximum delay. The black line represents the pareto front.

and synthesized targeting a standard-cell library as detailed in Section V.

The approximate quotient ϕ_k is computed by multiplying $c_{\text{int,k}}$ and $b_{\text{int,k}}$ with the mantissas and by adding $a_{\text{int,k}}$ to the products. With the aim to reduce the complexity of multipliers, nt LSBs of mantissas are truncated, obtaining the signals Mx_{nt} and My_{nt}. We underline that nt can be carefully chosen in dependence on the used floating-point format, and, accordingly, in dependence on the desired precision.

Moreover, the multipliers and the adder are organized in a fused carry-save arithmetic structure, named CSAS in the figure, to further optimize hardware.

The Figure 5b shows details of the CSAS in the case $N = 8$, $LSBa = 2^{-7}$, $LSBb = 2^{-1}$, $LSBc = 2^{-4}$ and $nt = 16$. Here, $a_{\text{int,k}}$, $|b_{\text{int,k}}|$ and $c_{\text{int,k}}$ are expressed on 8, 2 and 4 bits, respectively, whereas Mx_{nt}, My_{nt} are on $23 - nt = 7$ bits. Then, the first 4 blue rows are due to $Mx_{\text{nt}} \cdot c_{\text{int,k}}$, whereas the other 2 orange rows are related to $My_{\text{nt}} \cdot |b_{\text{int,k}}|$. The term $a_{\text{int,k}}$ is depicted in green. In addition, having Mx_{nt}, My_{nt} a LSB of weight $2^{-(23-nt)} = 2^{-7}$, the products $Mx_{\text{nt}} \cdot c_{\text{int,k}}$, $My_{\text{nt}} \cdot b_{\text{int,k}}$ have LSBs of weight 2^{-11} and 2^{-8}, respectively. It is also worth noting that the CSAS computes only 12 bits of the quotient instead of 24, thus allowing to reduce the hardware complexity of the normalization process (detailed in the following). In general, the number of bits computed by CSAS is $n_\phi = 24 - nt + |\log_2(LSBc)|$.

Finally, the Normalization block in Fig. 5 rearranges ϕ_k in the interval [1, 2) to extract the mantissa Mz. As stated in Section II, the quotient varies in [0.5, 2), and, accordingly, its MSB (indicated as $\phi_k[n_\phi - 1]$ in the figure) has a weight 2^0. If $\phi_k[n_\phi - 1] = 0$, then ϕ_k is in the range [0.5, 1) and the normalization process provides to add a zero at the least significant position in order to double the quotient (see the signal ϕ_1 in the Normalization block). Moreover, $\sim\phi_k[n_\phi - 1]$ is subtracted to the exponent for compensation, with "$\sim$" representing the inversion operator.

Conversely, if $\phi_k[n_\phi - 1] = 1$, then ϕ_k is already in [1, 2) and no further operation is required. In this case, the fractional part of ϕ_k corresponds to Mz (see the signal ϕ_2 in the figure). In the architecture of Fig. 5, a multiplexer selects between ϕ_1 and ϕ_2, and the result is expressed on 23 bits by adding zeros at the least significant position.

V. Assessment of Performances

A. Error Metrics

Let us indicate the exact and the approximate quotients as Q and Q_{apprx}, respectively. We define the approximation error $E = Q - Q_{\text{apprx}}$, while the Relative Error Distance and the

TABLE IV

COEFFICIENTS FOR $N = 32$, $LSB_A = 2^{-7}$, $LSB_B = 2^{-1}$, AND $LSB_C = 2^{-5}$

h*=5 MSBs of *My	$a_{int,k}$	$b_{int,k}$	$c_{int,k}$
00000	130	-3	31
00001	128	-2	31
00010	133	-3	30
00011	129	-2	29
00100	130	-2	28
00101	132	-2	27
00110	132	-2	27
00111	134	-2	26
01000	136	-2	25
01001	136	-2	25
01010	139	-2	24
01011	139	-2	24
01100	117	-1	23
01101	143	-2	23
01110	117	-1	22
01111	117	-1	22
10000	118	-1	21
10001	117	-1	21
10010	119	-1	20
10011	118	-1	20
10100	118	-1	20
10101	120	-1	19
10110	120	-1	19
10111	120	-1	19
11000	122	-1	18
11001	122	-1	18
11010	122	-1	18
11011	124	-1	17
11100	125	-1	17
11101	125	-1	17
11110	127	-1	16
11111	128	-1	16

Mean Relative Error Distance are $RED = |E/Q|$ and $MRED = avg(RED)$ as shown in Section II, where $avg(\cdot)$ is the average operator. We also compute the Error Bias defined as $EB = avg(E/Q)$ [49], and the probability of having *RED* larger than 2% (referred as *PRED* in the following).

The error metrics are computed by performing 10^6 divisions, with 10^6 couples of random, uniformly distributed, floating-point single-precision numbers.

In the following, we consider the cases (i), (ii), (iii), and (iv) presented in Section III-C for realizing the mantissas division, and name the corresponding floating-point dividers $FPDME_4(7, 1, 3)$, $FPDME_8(7, 1, 4)$, $FPDME_{16}(7, 1, 4)$, and $FPDME_{32}(7, 1, 5)$, respectively. We also vary the number of discarded LSBs *nt* and report the case without truncation for reference.

For the sake of comparison, the performances of dividers [42], [44], [48], [49], and [50] are also shown. The divider [42], named ALD in the following, subtracts mantissas in the LNS representation, processing only the first q MSB of Mx and My, with $q = 8$ in our trials. The work [49]

TABLE V

ERROR METRICS OF THE PROPOSED DIVIDER AND THE STATE-OF-THE-ART

Floating-point divider		***MRED***	***EB***	***PRED* (>2%)**
ALD, *q*=8 [42]		4.07%	4.07%	6.55×10^{-1}
LPCAD(2,4) [49]		2.23%	-1.63%	4.68×10^{-1}
LPCAD(2,8) [49]		1.30%	-0.10%	2.21×10^{-1}
LPCAD(3,4) [49]		1.94%	-1.73%	4.30×10^{-1}
LPCAD(3,8) [49]		0.75%	0.08%	8.28×10^{-2}
CADE *P*=3, *L*=8 [50]		1.30%	0.08%	2.19×10^{-1}
CADE *P*=4, *L*=8 [50]		0.65%	0.02%	1.81×10^{-2}
TruncApp *r*=4 [44]		4.20%	-0.84%	7.11×10^{-1}
FPAD L3A2 [48]		3.05%	1.59%	5.82×10^{-1}
FPAD L4A2 [48]		2.22%	-0.72%	4.44×10^{-1}
$FPDME_4(7,1,3)$	**no trunc.**	1.41%	0%	2.21×10^{-1}
	***nt*=15**	1.41%	0.01%	2.22×10^{-1}
	***nt*=17**	1.48%	0.05%	2.37×10^{-1}
$FPDME_8(7,1,4)$	**no trunc.**	0.77%	0.01%	7.95×10^{-2}
	***nt*=14**	0.78%	0.01%	8.03×10^{-2}
	***nt*=16**	0.81%	0.03%	8.51×10^{-2}
$FPDME_{16}(7,1,4)$	**no trunc.**	0.57%	0.05%	2.39×10^{-2}
	***nt*=14**	0.57%	0.06%	2.43×10^{-2}
	***nt*=16**	0.60%	0.06%	2.81×10^{-2}
$FPDME_{32}(7,1,5)$	**no trunc.**	0.33%	0.05%	3.18×10^{-4}
	***nt*=14**	0.33%	0.05%	3.69×10^{-4}
	***nt*=16**	0.38%	0.06%	4.85×10^{-4}

approximates $1/(1 + My)$ using 2^d values, with d that is 2 or 3, and exploits a truncated multiplier with t preserved columns. In the following, the divider [49] will be presented as LPCAD(d, t), with $t = 4, 8$. The work [50], named CADE in the following, divides the mantissas' plane in $2^P \times 2^P$ square regions and computes, for each section, an error compensation term expressed on L bits. For our study, we consider $L = 8$ and $P = 3, 4$. The design [44], referred as TruncApp, exploits linear approximation for the term $1/(1 + My)$, and employes only r bits for computing the quotient, with $r = 4$ in our trials. Finally, the work [48] involves 2^α possible shift-and-add operations for realizing the division, with α defining the approximation level. Moreover, each operation involves β adders, whose addends are truncated on 5 bits. In the following, we refer to [48] as FPAD LαAβ.

Table V collects the error metrics for both the proposed divider and the state-of-the-art, with *MRED* and *EB* reported in percentage values. As expected, the performance of the architecture proposed in this paper depends on the number of partitions N, with the *MRED* improving from 1.5% (for $N = 4$) to 0.33% (for $N = 32$). *PRED* also exhibits a marked dependance, passing from 2.4×10^{-1} to 3.2×10^{-4}, whereas *EB* results almost constant. In addition, also *nt* has an effect on the accuracy of the divider, with best approximation achieved when the number of truncated LSBs is low.

As for the other implementations, only LPCAD(2, 8), LPCAD(3, 8), and CADE are able to offer error metrics comparable with the proposed FPDME, with CADE $P = 4$,

TABLE VI
HARDWARE PERFORMANCES OF THE PROPOSED DIVIDER AND THE STATE-OF-THE-ART SYNTHESIS WITH MINIMUM AREA AND POWER

Floating-point divider		Area [μm²]	Delay [ns]	Power [μW@100MHz]	PDP [fJ]	ADP [μm²·ns]
Exact divider		2156.1	9.180	3098.5	28444.2	19793.1
ALD, *q*=8 [42]		44.1	0.508	6.2	3.1	22.4
LPCAD(2,4) [49]		83.2	0.942	13.9	13.1	78.3
LPCAD(2,8) [49]		116.0	0.948	20.0	18.9	110.0
LPCAD(3,4) [49]		79.1	0.862	15.8	13.6	83.7
LPCAD(3,8) [49]		139.4	0.988	23.2	23.0	137.7
CADE *P*=3, *L*=8 [50]		155.4	0.990	23.0	22.8	153.8
CADE *P*=4, *L*=8 [50]		293.7	0.990	37.9	37.6	290.8
TruncApp *r*=4 [44]		57.6	0.373	7.8	2.9	21.5
FPAD L3A2 [48]		102.69	0.808	14.3	11.6	83.0
FPAD L4A2 [48]		113.27	0.870	15.9	13.9	98.5
FPDME$_4$(7,1,3)	***nt*=15**	113.3	0.704	17.3	12.9	79.7
	***nt*=17**	92.5	0.658	14.3	9.4	60.9
FPDME$_8$(7,1,4)	***nt*=16**	142.3	0.856	24.5	21.0	121.8
FPDME$_{16}$(7,1,4)	***nt*=16**	142.0	0.914	26.1	23.9	129.8
FPDME$_{32}$(7,1,5)	***nt*=16**	167.8	0.938	30.2	28.3	157.4

$L = 8$ that achieves *MRED* of 0.65%. The other dividers exhibit poorer accuracy, with *MRED* in the order of 2% or larger. In this case, ALD and TruncApp show worst results, with *MRED* around 4% and *PRED* of about 7×10^{-1}.

B. Hardware Performances

We have described the proposed dividers and the state-of-the-art dividers in Verilog HDL and synthesized the circuits in TSMC 28nm CMOS technology using a physical flow in Cadence Genus.

For the proposed FPDME architecture, we have implemented FPDME$_4$(7, 1, 3) with either $nt = 15$, and $nt = 17$, while FPDME$_8$(7, 1, 4), FPDME$_{16}$(7, 1, 4), and FPDME$_{32}$(7, 1, 5) have been implemented with $nt = 16$. As mentioned, the LUTs are described by means of procedural blocks and are implemented during the synthesis process with the standard cells of the library.

In a first experiment, we have imposed a very loose constraint on the maximum delay of the circuits (10ns), so that the synthesizer is able to implement minimum area and minimum power versions of the dividers. In this case we also synthesized the exact floating-point divider, chosen from the ChipAware library of the synthesizer.

In a second experiment, we have imposed a tighter constraint on the maximum delay (750ps), to investigate the performance when a higher operating frequency is required. In this second experiment, we have opted to exclude the exact divider due to the complexity of the circuit, making it impractical to meet the timing constraint.

In both experiments the power consumption is obtained by simulating the synthesized netlists with 10^5 random inputs, with path delays annotated in standard delay format (SDF) file and switching activity annotated in toggle count format (TCF) file.

Table VI collects the results for the first experiment. The last two columns report the power-delay product (PDP) and the area-delay product (ADP).

All the investigated architectures drastically reduce the PDP with respect to the exact divider. Best results are shown by ALD and TruncApp, with PDP in the order of 3fJ. These architectures, however, are also the one with the largest error. The proposed architecture shows good tradeoff between error and PDP. For instance, FPDME$_4$(7, 1, 3) $nt = 17$ exhibits a lower PDP compared to all versions of LPCAD, CADE, and FPAD, and it also has lower error (with the sole exception of CADE $P = 4$ $L = 8$, LPCAD(2, 8) and LPCAD(3, 8)). A similar behavior is also shown for the ADP.

Likewise, Table VII collects results for the second experiment. As shown, our dividers offer PDP and ADP comparable to LPCAD, CADE $P = 3$, $L = 8$, and FPAD, with best results achieved by FPDME$_4$(7, 1, 3) $nt = 17$. ALD and TruncApp show best hardware complexity, whereas CADE $P = 4$, $L = 8$ exhibits worse PDP and ADP.

In order to have a joint assessment of electrical and accuracy performances, Fig. 6 depicts the PDP and the ADP with respect to the *MRED* for both experiments. Here, implementations closer to the bottom-left corner exhibit low PDP/ADP with high accuracy, thus defining the Pareto front.

As shown in Fig. 6a, the proposed dividers offer the best trade-off between PDP and *MRED* and are all on the Pareto front (highlighted by the black dashed line). Only LPCAD(3, 8) is close to the optimal curve, whereas other implementations show worse behaviors, with the only exception of ALD and TruncApp showing, however, a large *MRED*. The proposed FPDME are on the pareto front also in Fig. 6b, determining the best trade-off between ADP and *MRED*. Again LPCAD(3, 8) results competitive as well as ALD and TruncApp for low accuracy. A similar trend is shown also in Fig. 6c and 6b for

TABLE VII
HARDWARE PERFORMANCES OF THE PROPOSED DIVIDER AND THE STATE-OF-THE-ART SYNTHESIS WITH A 750ps CONSTRAINT ON THE MAXIMUM DELAY

Floating-point divider		Area [μm²]	Delay [ns]	Power [μW@1.33GHz]	PDP [fJ]	ADP [μm²·ns]
ALD, *q*=8 [42]		46.1	0.489	79.0	38.6	22.6
LPCAD(2,4) [49]		90.0	0.583	180.6	105.3	52.4
LPCAD(2,8) [49]		122.6	0.686	267.4	183.5	84.1
LPCAD(3,4) [49]		103.7	0.622	211.3	131.4	64.5
LPCAD(3,8) [49]		145.3	0.720	302.2	217.6	104.6
CADE *P*=3, *L*=8 [50]		172.9	0.731	305.1	223.1	126.4
CADE *P*=4, *L*=8 [50]		310.7	0.750	467.4	350.5	233.0
TruncApp *r*=4 [44]		57.6	0.373	98.9	36.9	21.5
FPAD L3A2 [48]		106.1	0.728	191.4	139.3	77.2
FPAD L4A2 [48]		117.9	0.749	211.4	158.3	88.3
FPDME$_4$(7,1,3)	***nt*=15**	113.4	0.705	228.7	161.2	79.9
	***nt*=17**	92.2	0.653	185.9	121.4	60.2
FPDME$_8$(7,1,4)	***nt*=16**	145.8	0.746	299.7	223.6	108.8
FPDME$_{16}$(7,1,4)	***nt*=16**	152.8	0.743	365.1	271.2	113.6
FPDME$_{32}$(7,1,5)	***nt*=16**	182.8	0.748	411.9	308.1	136.8

TABLE VIII
PERFORMANCES OF THE PROPOSED DIVIDER AND THE STATE-OF-THE-ART IN CHANGE DETECTION APPLICATION

Floating-point divider		Walter Cronkite		Chemical Plant (far view)		Chemical Plant (close view)		Toy Vehicle		Average	
		PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
ALD, *q*=8 [42]		36.0	0.983	33.5	0.987	31.3	0.985	44.7	0.999	36.4	0.989
LPCAD(2,4) [49]		33.7	0.932	35.9	0.988	33.6	0.988	36.6	0.884	34.9	0.948
LPCAD(2,8) [49]		42.4	0.988	41.2	0.997	38.3	0.994	52.3	0.999	43.5	0.994
LPCAD(3,4) [49]		33.2	0.917	36.9	0.989	34.6	0.990	35.7	0.859	35.1	0.939
LPCAD(3,8) [49]		47.3	0.996	46.3	0.999	43.5	0.998	54.8	0.999	48.0	0.998
CADE *P*=3, *L*=8 [50]		42.6	0.982	43.1	0.997	40.1	0.995	43.1	0.972	42.2	0.986
CADE *P*=4, *L*=8 [50]		49.9	0.996	49.6	0.999	46.7	0.999	49.8	0.993	49.0	0.997
TruncApp *r*=4 [44]		30.6	0.863	32.4	0.967	29.9	0.957	36.9	0.967	32.5	0.938
FPAD L3A2 [48]		35.9	0.938	35.1	0.982	32.3	0.974	37.1	0.938	35.1	0.958
FPAD L4A2 [48]		35.0	0.932	37.3	0.987	25.0	0.984	36.7	0.936	36.0	0.960
FPDME$_4$(7,1,3)	***nt*=15**	40.1	0.978	39.9	0.994	38.3	0.993	40.0	0.980	39.6	0.986
	***nt*=17**	40.1	0.978	39.9	0.994	38.3	0.993	39.8	0.974	39.5	0.985
FPDME$_8$(7,1,4)	***nt*=16**	45.4	0.992	46.6	0.999	44.0	0.998	50.0	0.994	46.5	0.996
FPDME$_{16}$(7,1,4)	***nt*=16**	48.3	0.995	49.4	0.999	47.0	0.999	53.9	0.997	49.6	0.998
FPDME$_{32}$(7,1,5)	***nt*=16**	51.9	0.998	53.0	1.000	50.7	1.000	58.7	0.999	53.6	0.999

the faster implementations, where the proposed dividers define or are very close to the pareto front.

VI. APPLICATIONS

A. Change Detection

Change detection is often employed in computer vision to highlight motion in subsequent frames. The division between pixels is suitable to detect differences between images. Indeed, if objects do not move, their pixels are practically constant among the frames and, accordingly, their division is very close to 1. Conversely, division is far from 1 in case of a change, thus highlighting a motion.

In this paragraph, we analyze the performances of the proposed divider and the state-of-the-art when changes are detected in the frames Walter Cronkite, Chemical Plant (far and close view), and Toy Vehicle, from the database [52]. For our assessments, we report the peak signal-to-noise ratio (PSNR), expressed in dB, and the mean structural similarity index (SSIM), commonly used to qualify algorithms in image and video processing. In addition, we also report for each investigated divider the average PSNR and the average SSIM among the four experiments.

TABLE IX
PERFORMANCES OF THE PROPOSED DIVIDER AND THE STATE-OF-THE-ART IN JPEG COMPRESSION

Floating-point divider		Q=40		Q=70		Q=100		Average	
		PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
ALD, *q*=8 [42]		35.7	0.960	36.3	0.963	36.4	0.969	36.1	0.964
LPCAD(2,4) [49]		41.8	0.989	42.7	0.992	45.8	0.997	43.4	0.992
LPCAD(2,8) [49]		44.3	0.993	45.6	0.995	55.4	0.999	48.4	0.996
LPCAD(3,4) [49]		42.1	0.989	43.3	0.993	45.8	0.997	43.7	0.993
LPCAD(3,8) [49]		44.9	0.993	46.8	0.996	56.4	0.999	49.3	0.996
CADE *P*=3, *L*=8 [50]		41.4	0.987	45.0	0.994	49.2	0.997	45.2	0.993
CADE *P*=4, *L*=8 [50]		46.1	0.994	47.7	0.996	52.5	0.998	48.8	0.996
TruncApp *r*=4 [44]		38.5	0.975	39.1	0.979	42.5	0.989	40.0	0.981
FPAD L3A2 [48]		40.5	0.986	44.4	0.993	39.1	0.988	41.3	0.989
FPAD L4A2 [48]		40.8	0.986	44.6	0.993	48.7	0.997	44.7	0.992
FPDME$_4$(7,1,3)	***nt*=15**	43.3	0.993	44.1	0.992	53.3	0.999	46.9	0.995
	***nt*=17**	43.9	0.993	43.8	0.992	50.9	0.998	46.2	0.995
FPDME$_8$(7,1,4)	***nt*=16**	43.9	0.991	45.4	0.996	51.4	0.998	46.9	0.995
FPDME$_{16}$(7,1,4)	***nt*=16**	44.5	0.993	47.5	0.996	55.1	0.999	49.1	0.996
FPDME$_{32}$(7,1,5)	***nt*=16**	44.8	0.993	48.6	0.997	57.8	0.999	50.4	0.996

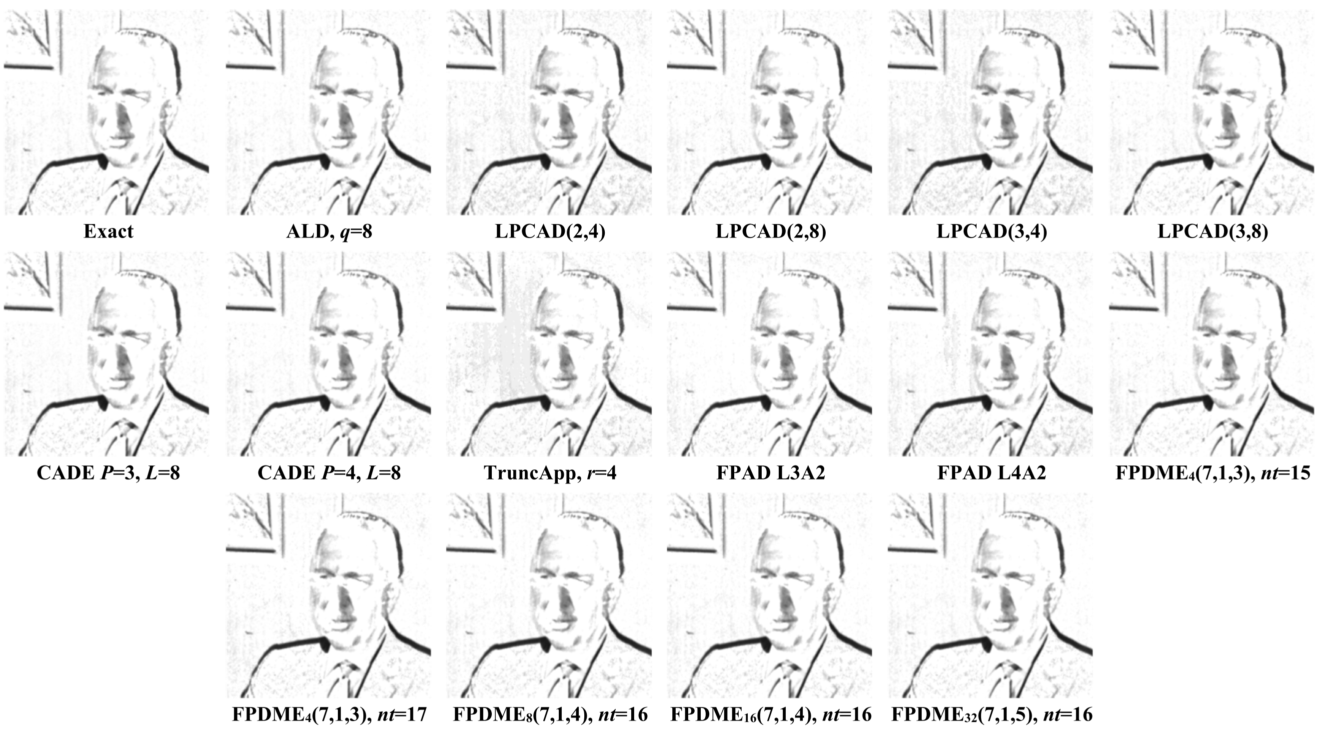


Fig. 7. Change detection for Walter Cronkite image with proposed dividers and the state-of-the-art.

As shown in Table VIII, our proposal is competitive with the state-of-the-art and offers remarkable results, with SSIM very close to 1 in all the cases, and average PSNR in the range 46.5dB/53.6dB for $N \geq 8$. In addition, FPDME$_{32}$(7, 1, 5) overcomes 50dB in all the trials and achieves the highest PSNR (58.7dB) with Toy Vehicle. The implementations ALD, TruncApp and FPAD show poorer performances, with an average PSNR lower than 40dB and an average SSIM of 0.938 in the case of TruncApp. Accuracy of LPCAD depends on the approximation parameters, with PSNR varying between 35dB and 48dB, whereas CADE performs better, showing PSNR slightly less than 50dB. Figure 7 represents the image obtained by dividing the frames of Walter Cronkite. As shown, results obtained with LPCAD(2, 4), LPCAD(3, 4), and TruncApp exhibit a visible degradation in the background, whereas the proposed dividers allow to get images practically unchanged with respect to the exact case.

B. JPEG Compression

As further example, we assess the accuracy of approximate dividers in JPEG image compression. The JPEG compression exploits cosine transformation and variable quantization to approximate images. The compression algorithm roughly

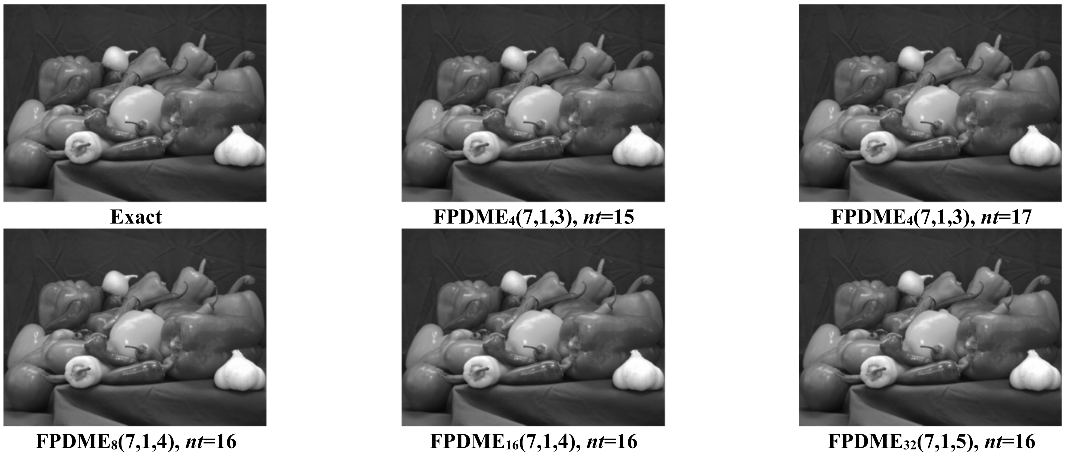


Fig. 8. JPEG compressions in the case of Peppers image with the exact and the proposed dividers, $Q = 100$.

quantizes the high frequencies of an image, whereas employes a finer quantization step to approximate the low frequencies. In this way, the compression algorithm reduces the size of images in memory at the cost of a worse representation of the high frequencies, which are less evident to human eye. In addition, an approximation factor Q, laying in the range [0, 100], allows to define the overall amount of compression by modifying the quantization steps, with $Q = 0$ and $Q = 100$ indicating worst and finest quantization, respectively. In our case, we employ the approximate dividers in the quantization phase since a division between the pixels and the variable quantization steps is required. Three test images, Lena, Cameraman, and Peppers, are considered for our simulations, compressed with factors $Q = 40$, $Q = 70$ and $Q = 100$. For each Q and for each image, we compute the PSNR and SSIM comparing the approximate and the exact results. Then, we average the PSNR and SSIM computed for each Q, reporting the respective values in Table IX. In addition, the overall average PSNR and SSIM are also shown in the last two columns of the table.

As observable, the proposed dividers are competitive with the state-of-the-art, exhibiting both high values of PSNR and SSIM. Best results are achieved in the case $Q = 100$, with PSNR larger than 55dB for $N = 16, 32$. In addition, $\text{FPDME}_{32}(7, 1, 5)$ is the only one able to achieve an average PSNR of 50dB. Figure 8 confirms these observations since the images compressed with the proposed and the exact dividers are practically undistinguishable.

Among the other implementations, only LPCAD(3, 8) and CADE $P = 4$, $L = 8$ offer results comparable to $\text{FPDME}_{16}(7, 1, 4)$ and $\text{FPDME}_{32}(7, 1, 5)$, with PSNR around 48dB/49dB and SSIM very close to 1. FPAD L4A2 achieves 44.7dB PSNR, whereas ALD exhibits worst performances, with average PSNR of 36dB and average SSIM of 0.964.

VII. Conclusion

In this paper, we have proposed a novel non-iterative approximate floating-point divider based on linear approximation.

In our divider, we have approximated the quotient $(1 + Mx)/(1 + My)$ as a linear function of Mx with coefficients dependent on My. The coefficients have been calculated to minimize the Mean Relative Error Distance (*MRED*) of the approximation. To this end, the range of My has been partitioned in N sub-intervals and in each sub-interval the minimization of *MRED* has been formulated as a linear programming problem, whose solution gives optimal coefficient values. Mantissa truncation and coefficient quantization have also been exploited to further optimize the design.

The hardware structure of the whole floating-point divider has been described in detail, and the performance of the proposed architecture has been compared with previously proposed approximate dividers. Our analysis shows that the proposed architecture overcomes the state of the art, offering the best trade-off between PDP/ADP and accuracy for a wide range of mean relative error distance values. We have also presented results for two image processing applications that both remark the advantages of the proposed technique, exhibiting competitive performances in terms of peak signal-to-noise ratio (PSNR) and Mean Structural Similarity Index (SSIM).

References

[1] 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.

[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013, doi: 10.1016/j.future.2013.01.010.

[3] F. Spagnolo, S. Perri, and P. Corsonello, "Approximate down-sampling strategy for power-constrained intelligent systems," *IEEE Access*, vol. 10, pp. 7073–7081, 2022, doi: 10.1109/ACCESS.2022.3142292.

[4] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proc. 18th IEEE Eur. Test Symp. (ETS)*, Avignon, France, May 2013, pp. 1–6, doi: 10.1109/ETS.2013.6569370.

[5] 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 Design Autom. Conf. (DAC)*, Austin, TX, USA, May 2013, pp. 1–9, doi: 10.1145/2463209.2488873.

[6] R. J. Radke, S. Andra, O. Al-Kofahi, and B. Roysam, "Image change detection algorithms: A systematic survey," *IEEE Trans. Image Process.*, vol. 14, no. 3, pp. 294–307, Mar. 2005, doi: 10.1109/TIP.2004.838698.

[7] D. Esposito, G. Di Meo, D. De Caro, A. G. M. Strollo, and E. Napoli, "Quality-scalable approximate LMS filter," in *Proc. 25th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Bordeaux, France, Dec. 2018, pp. 849–852, doi: 10.1109/ICECS.2018.8617858.

[8] G. Di Meo, D. De Caro, G. Saggese, E. Napoli, N. Petra, and A. G. M. Strollo, "A novel module-sign low-power implementation for the DLMS adaptive filter with low steady-state error," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 1, pp. 297–308, Jan. 2022, doi: 10.1109/TCSI.2021.3088913.

[9] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, "Design of voltage-scalable meta-functions for approximate computing," in *Proc. Design, Automat. Test Europe*, Grenoble, France, 2011, pp. 1–6, doi: 10.1109/DATE.2011.5763154.

[10] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Proc. Design Autom. Conf.*, San Francisco, CA, USA, Jun. 2012, pp. 820–825, doi: 10.1145/2228360.2228509.

[11] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, Jun. 2015, pp. 1–6, doi: 10.1145/2744769.2744778.

[12] K. Du, P. Varman, and K. Mohanram, "High performance reliable variable latency carry select addition," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Dresden, Germany, Mar. 2012, pp. 1257–1262, doi: 10.1109/DATE.2012.6176685.

[13] Y. Kim, Y. Zhang, and P. Li, "An energy efficient approximate adder with carry skip for error resilient neuromorphic VLSI systems," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, Nov. 2013, pp. 130–137, doi: 10.1109/ICCAD.2013.6691108.

[14] L. Li and H. Zhou, "On error modeling and analysis of approximate adders," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, Nov. 2014, pp. 511–518, doi: 10.1109/ICCAD.2014.7001399.

[15] D. Esposito, D. De Caro, and A. G. M. Strollo, "Variable latency speculative parallel prefix adders for unsigned and signed operands," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 8, pp. 1200–1209, Aug. 2016, doi: 10.1109/TCSI.2016.2564699.

[16] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 4, pp. 850–862, Apr. 2010, doi: 10.1109/TCSI.2009.2027626.

[17] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi, "Approximate XOR/XNOR-based adders for inexact computing," in *Proc. 13th IEEE Int. Conf. Nanotechnol.*, Beijing, China, Aug. 2013, pp. 690–693, doi: 10.1109/NANO.2013.6720793.

[18] 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.

[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)*, Amherst, MA, USA, Oct. 2015, pp. 183–186, doi: 10.1109/DFT.2015.7315159.

[20] 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, doi: 10.1109/LES.2017.2746084.

[21] 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, doi: 10.1109/TVLSI.2016.2643003.

[22] 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.

[23] N. Petra, D. De Caro, V. Garofalo, E. Napoli, and A. G. M. Strollo, "Truncated binary multipliers with variable correction and minimum mean square error," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 6, pp. 1312–1325, Jun. 2010, doi: 10.1109/TCSI.2009.2033536.

[24] J. M. Jou, S. R. Kuang, and R. Der Chen, "Design of low-error fixed-width multipliers for DSP applications," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 46, no. 6, pp. 836–842, Jun. 1999, doi: 10.1109/82.769795.

[25] 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, doi: 10.1109/TVLSI.2014.2333366.

[26] 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.

[27] G. Di Meo, G. Saggese, A. G. M. Strollo, and D. De Caro, "Design of generalized enhanced static segment multiplier with minimum mean square error for uniform and nonuniform input distributions," *Electronics*, vol. 12, p. 446, Jan. 2023, doi: 10.3390/electronics12020446.

[28] 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, doi: 10.1109/ICCAD.2015.7372600.

[29] 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, doi: 10.1109/TVLSI.2018.2890712.

[30] N. Burgess and C. N. Hinds, "Design of the ARM VFP11 divide and square root synthesisable macrocell," in *Proc. 18th IEEE Symp. Comput. Arithmetic (ARITH)*, Jun. 2007, pp. 87–96.

[31] G. Gerwig, H. Wetter, E. M. Schwarz, and J. Haess, "High performance floating-point unit with 116 bit wide divider," in *Proc. 16th IEEE Symp. Comput. Arithmetic*, Mar. 2003, pp. 87–94.

[32] S. F. Oberman, "Floating point division and square root algorithms and implementation in the AMD-K7^{TM} microprocessor," in *Proc. 14th IEEE Symp. Comput. Arithmetic*, Apr. 1999, pp. 106–115.

[33] D. W. Sweeney, "Divider device for skipping a string of zeros or radix-minus-one digits," U.S. Patent 3 145 296, Aug. 18, 1964.

[34] J. E. Robertson, "A new class of digital division methods," *IRE Trans. Electron. Comput.*, vol. EC-7, no. 3, pp. 218–222, Sep. 1958, doi: 10.1109/TEC.1958.5222579.

[35] K. D. Tocher, "Techniques of multiplication and division for automatic binary computers," *Quart. J. Mech. Appl. Math.*, vol. 11, no. 3, pp. 364–384, 1958, doi: 10.1093/qjmam/11.3.364.

[36] M. J. Flynn, "On division by functional iteration," *IEEE Trans. Comput.*, vol. C-19, no. 8, pp. 702–706, Aug. 1970, doi: 10.1109/T-C.1970.223019.

[37] R. E. Goldschmidt, "Applications of division by convergence," Ph.D. dissertation, Massachusetts Inst. Technol., Cambridge, MA, USA, 1964.

[38] J. Ebergen and N. Jamadagni, "Radix-2 division algorithms with an over-redundant digit set," *IEEE Trans. Comput.*, vol. 64, no. 9, pp. 2652–2663, Sep. 2015, doi: 10.1109/TC.2014.2366738.

[39] L. Chen, J. Han, W. Liu, and F. Lombardi, "Design of approximate unsigned integer non-restoring divider for inexact computing," in *Proc. 25th Great Lakes Symp. VLSI*, May 2015, pp. 51–56.

[40] L. Chen, J. Han, W. Liu, and F. Lombardi, "On the design of approximate restoring dividers for error-tolerant applications," *IEEE Trans. Comput.*, vol. 65, no. 8, pp. 2522–2533, Aug. 2016, doi: 10.1109/TC.2015.2494005.

[41] S. Hashemi, R. I. Bahar, and S. Reda, "A low-power dynamic divider for approximate applications," in *Proc. 53rd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Austin, TX, USA, Jun. 2016, pp. 1–6, doi: 10.1145/2897937.2897965.

[42] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Trans. Electron. Comput.*, vol. EC-11, no. 4, pp. 512–517, Aug. 1962, doi: 10.1109/TEC.1962.5219391.

[43] R. Zendegani, M. Kamal, A. Fayyazi, A. Afzali-Kusha, S. Safari, and M. Pedram, "SEERAD: A high speed yet energy-efficient rounding-based approximate divider," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Dresden, Germany, Mar. 2016, pp. 1481–1484.

[44] S. Vahdat, M. Kamal, A. Afzali-Kusha, M. Pedram, and Z. Navabi, "TruncApp: A truncation-based approximate divider for energy efficient DSP applications," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, Lausanne, Switzerland, Mar. 2017, pp. 1635–1638, doi: 10.23919/DATE.2017.7927254.

[45] H. Saadat, H. Javaid, and S. Parameswaran, "Approximate integer and floating-point dividers with near-zero error bias," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, Las Vegas, NV, USA, Jun. 2019, pp. 1–6.

[46] M. Vaeztourshizi, M. Kamal, A. Afzali-Kusha, and M. Pedram, "An energy-efficient, yet highly-accurate, approximate non-iterative divider," in *Proc. Int. Symp. Low Power Electron. Design*, New York, NY, USA, Jul. 2018, pp. 1–6, doi: 10.1145/3218603.3218650.

[47] *IEEE Standard for Floating-Point Arithmetic*, IEEE Standard 754-2019, Jul. 2019, doi: 10.1109/IEEESTD.2019.8766229.

[48] C. K. Jha, K. Prasad, V. K. Srivastava, and J. Mekie, "FPAD: A multi-stage approximation methodology for designing floating point approximate dividers," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Seville, Spain, Oct. 2020, pp. 1–5, doi: 10.1109/ISCAS45731.2020.9180768.

[49] Y. Wu et al., "An energy-efficient approximate divider based on logarithmic conversion and piecewise constant approximation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 7, pp. 2655–2668, Jul. 2022, doi: 10.1109/TCSI.2022.3167894.

[50] M. Imani, R. Garcia, A. Huang, and T. Rosing, "CADE: Configurable approximate divider for energy efficiency," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Florence, Italy, Mar. 2019, pp. 586–589, doi: 10.23919/DATE.2019.8715112.

[51] L. Wu and C. C. Jong, "A curve fitting approach for non-iterative divider design with accuracy and performance trade-off," in *Proc. IEEE 13th Int. New Circuits Syst. Conf. (NEWCAS)*, Grenoble, France, Jun. 2015, pp. 1–4, doi: 10.1109/NEWCAS.2015.7182097.

[52] *The USC-SIPI Image Database*. [Online]. Available: https://sipi.usc.edu/database/

Gennaro Di Meo received the M.S. degree (cum laude) in electrical engineering and the Ph.D. degree in information technology and electrical engineering from the University of Naples Federico II, Italy, in 2018 and 2022, respectively. He is currently a Post-Doctoral Researcher with the Department of Information Technology and Electrical Engineering, University of Naples Federico II. His research interests include design of digital VLSI circuits for telecommunications, LSM filters, and approximate computing.

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 papers 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 of international conferences, including PRIME, ICECS, and ESSCIRC/ESSDERC. He and his coauthors were a recipient of the 2021 IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS 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*.

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 with 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.

Open Access funding provided by 'Università degli Studi di Napoli "Federico II"' within the CRUI CARE Agreement