Low-Power High Precision Floating-Point Divider With Bidimensional Linear Approximation

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

***Abstract*— In this paper we propose a novel approximate floating-point divider based on bidimensional linear approximation. In our approach, the mantissa quotient is seen as a function of the two input mantissas of the divider. The domain of this two-variable function is partitioned into $nx \times ny$ subregions, named tiles, where nx, ny are chosen as powers of two. In each tile the quotient is approximated with a linear combination of the input mantissas. To achieve fine accuracy, an optimization problem is formulated within each tile to determine the optimal coefficients for the linear combination, which minimize the Mean Relative Error Distance (*MRED*) of the divider. Furthermore, to make hardware implementation more effective, the minimization problem is appropriately modified to search for optimal quantized coefficients. The hardware structure of the divider only requires a small look-up table to store the linear approximation coefficients, and a carry save adder tree. The proposed architecture is highly tunable at design-time over a wide range of accuracy, depending on the number of tiles chosen for the approximation. The obtained results demonstrate error performance and hardware features superior to the state-of-the-art. The proposed dividers define the Pareto front, considering the trade-off between power-delay-product vs. *MRED* and area-delay-product vs. *MRED*, for *MRED* in the range of $4 \times 10^{-3} - 2 \times 10^{-2}$. Application results for JPEG compression and tone mapping further highlight the strength of our proposal, which exhibits Structural Similarity Index (SSIM) very close to 1 in all cases and Peak Signal-to-Noise Ratio (PSNR) up to 45 dB.**

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

I. Introduction

ARITHMETIC circuits are widely employed for the design of Digital Signal Processing (DSP) algorithms, typically being the key elements to achieve the highest performances. The demand for suitable hardware features in terms of area occupation and power consumption is of pivotal importance to ensure the feasible integration of these circuits in electronic systems and makes their implementation challenging. Among applications requiring careful design of arithmetic circuits, environmental sensing and healthcare monitoring occupy an interesting position, involving the realization of complex networks of devices where data are acquired, processed, and transmitted according to the paradigm of the Internet of Things [1]. In this context, signal elaboration, leveraging addition, multiplication and division, is responsible for large energy consumption, thus calling for the development of suitable low-power techniques. Likewise, tasks as image classification or speech enhancement demand for the design of neural networks, exploiting additions and multiplications extensively [2]. Also in this case, power and area constitute critical design parameters that make the practical realization of the algorithms challenging. As further example, also in telecommunication systems circuits for adaptive filtering, channel equalization, Fast Fourier Transform (FFT) or Inverse Fast Fourier Transform (IFFT) [3], [4], [5] require an intensive usage of arithmetic blocks to realize noise suppression and signal coding. Again, the research of methos able to allow an effective integration of such systems is crucial for their practical realization, and, as consequence, the adoption of proper strategies for arithmetic circuits design becomes relevant to meet the desired power/area requirements.

A. General Background

In the field of DSP, floating-point arithmetic constitutes a valuable mean to perform calculations with fine accuracy in a wide range of representable values. At the same time, the practical realization of floating-point arithmetic circuits poses important issues from a hardware perspective, thus demanding for proper implementation strategies. In particular, the computation of the output mantissa, generally expressed on a large number of bits, implies intensive calculations that lead to high power consumption and area. In the context of signal processing, Approximate Computing constitutes a meaningful solution to reduce the complexity of the circuits, providing to intentionally sacrifice accuracy of calculation to save hardware performance [6], [7].

In literature, several approximate techniques have been shown for the design of efficient floating-point arithmetic circuits. In the case of multipliers, proposed approaches range from linear approximation [8], [9] to static segmentation [10]

Manuscript received 9 May 2024; revised 17 July 2024 and 16 August 2024; accepted 19 August 2024. This work was supported by the Spoke 1 Future High Performance Computing (HPC) and Big Data of Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC) funded by Ministero dell'Università e della Ricerca (MUR) Mission 4—Next Generation European Union (EU). This article was recommended by Associate Editor J. Zhang. *(Corresponding author: Gennaro Di Meo.)*

Gennaro Di Meo, Antonio Giuseppe Maria Strollo, Davide De Caro, and Luca Tegazzini are with DIETI, University of Napoli Federico II, 80125 Napoli, Italy (e-mail: gennaro.dimeo@unina.it; astrollo@unina.it; dadecaro@unina.it; luca.tegazzini@unina.it).

Ettore Napoli is with DIEM, University of Salerno, 84084 Fisciano, Italy (e-mail: enapoli@unisa.it).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2024.3447830.

Digital Object Identifier 10.1109/TCSI.2024.3447830

and tunable accuracy implementations [11], [12]. In [13], and [14], mantissa product is also skipped under proper error conditions. Likewise, important efforts have been made also to reduce the complexity of floating-point adders exploiting techniques like lower-OR addition (LOA) and speculation [15], [16], [17].

The floating-point divider, necessary in a wide range of applications such as domain transformation, JPEG compression, tone mapping, and motion detection, is a notorious complex, slow and power-hungry circuit. Many recent research efforts have been dedicated to the design of hardware efficient floating-point dividers. Iterative algorithms like Sweeney-Robertson-Tocher (SRT) and Newton-Raphson are often employed to realize a division [18], [19], [20]. While in the SRT method each iteration allows to find a bit of the result exploiting subtraction, in the Newton-Raphson approach the algorithm starts from an initial estimate of the quotient and uses multiplications to achieve the desired value. Despite the precision of these techniques, iterative implementations suffer from high latency, and require significant hardware resources to store the partial results. On the contrary, works like [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], and [31] introduce suitable approximate techniques which allow to realize the division with optimal latency and throughput. For instance, in [21], [22], and [23] the authors revisit the division as a subtraction by computing the result in the logarithmic number system. The papers [24], [25], [26] find the reciprocal of the divisor performing a multiplication instead of a division, while [27], [28], and [29] introduce error correction techniques based on variable compensation terms. The works [30], [31] present solutions based on signal inversion and truncation, which are applicable to the mantissa divider for a hardware efficient computation.

B. Contribution

In this paper, we propose a novel approximate floating-point divider based on a bidimensional linear approximation. In our implementation, the mantissa division is revisited as function of the two input mantissas. The domain of this two-variable function is divided in rectangular regions, named tiles in the following, and in each tile the quotient is expressed as a piecewise-linear combination of the mantissas. The coefficients for the linear combination are obtained in each tile by solving an optimization problem aimed at minimizing the Mean Relative Error Distance (*MRED*) between the exact division and the proposed approximation. In a subsequent step, to make hardware implementation more effective, the optimization problem is properly modified to find a set of quantized coefficients able to minimize *MRED* while being represented with a reduced number of bits. The hardware implementation of the proposed circuit only requires a small look-up table (LUT), used to store the optimal coefficients, and a simple multi-operand adder.

Simulation results reveal error metrics comparable to or superior to the state-of-the-art, with accuracy tunable at design-time over a wide range depending on the number of tiles chosen for the approximation. In addition, post-synthesis results in TSMC 28nm CMOS technology reveal remarkable hardware performance, assessed in terms of power-delay-product (*PDP*) and area-delay-product (*ADP*), making the proposed design pareto optimal. Finally, analyses of applications as JPEG compression and tone mapping [32] of high dynamic range (HDR) images are presented to test the proposed design in concrete scenarios. Again, our approach demonstrates superior performance, offering results very close to the exact case.

The paper is organized as follows: section II offers an overview of the state-of-the-art and introduces the division in floating-point arithmetic; section III describes the theoretical approach adopted to compute the quotient with the bidimensional linear approximation, whereas section IV describes the hardware structure of the proposed mantissa divider. Error metrics and hardware performance are shown in section V, whereas image processing results are discussed in section VI. Conclusions are in section VII.

II. Floating-Point Division Background

A. Related Works

In the following we present an overview of the state-of-the-art for approximate, non-iterative dividers. A shown in [21], mantissa division can be computed by performing a subtraction in the logarithmic number system. This approach approximates the logarithm of the input mantissas and the final antilogarithm operation, introducing a rather large error on the result. In addition, some least significant bits (LSBs) of the mantissas can be truncated to further reduce the hardware cost. The authors of [22] exploit mantissa subtraction, as in [21], and introduce a constant compensation term to achieve zero mean error, while [23] presents a pipelined structure for field programmable gate arrays applications. In [24], a two steps approximation process is exploited: in the first step different approximation levels, which depend on the most significant bits of the divisor, are used to compute the reciprocal of the divisor, while in the second step the reciprocal of the divisor is multiplied by the dividend using shift-and-add operations. The number of approximation levels and the number of adders used for the shift-and-add operations define both the accuracy and the circuital complexity of the divider. As in [24], an approximate version of the reciprocal of the divisor is computed also in [25], which exploits a variant of the Taylor expansion, and a variable correction term, stored in a LUT, is added to recover precision. In [26] the authors propose an approximate division scheme based on logarithmic conversion and piecewise constant approximation for the reciprocal of the divisor. In this case, different from [24] an inner truncated multiplier is implemented to compute the quotient, achieving higher accuracy at the cost of a superior hardware complexity. The work [27] is also based on logarithmic numbers but introduces a variable correction technique for accuracy recovery as in [25]. Here, N MSBs of the input mantissas allow to identify $2^{2 \cdot N}$ regions, and, for each region, a coefficient is defined to reduce the approximation error. Paper [28] also exploits a similar approach, with the mantissa plane divided in both rectangular and triangular regions. Accordingly, suitable

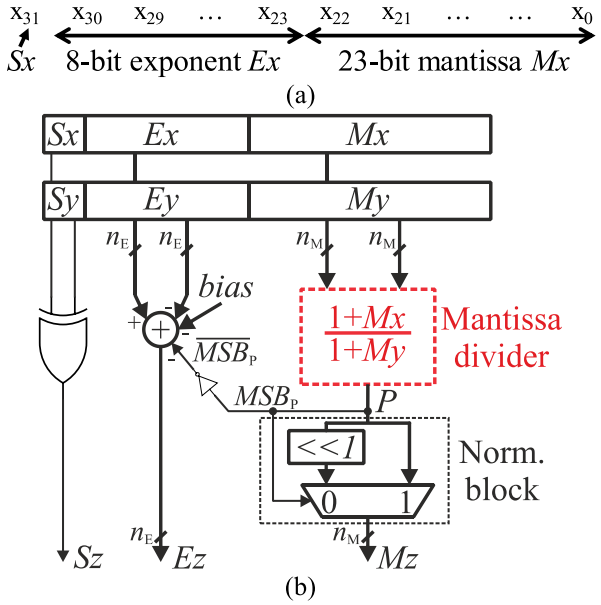


Fig. 1. (a) Floating-point representation of the real number x in single precision format, and (b) block diagram of the floating-point divider.

coefficients that best fit the division with a linear function of the mantissas are found. In [29], the quotient is expressed as a linear function of the dividend, with coefficients that depend on the divisor. Some LSBs of the operands are truncated to reduce the hardware burden. In the case of [27], [28], [29], the kind of partition scheme and the number of truncated bits are design parameters used to trade accuracy of calculations for hardware complexity. The work [30] approximates the reciprocal of the divisor with a linear function. Then, following an approach similar to [24], revisits the computed expression as a shift-and-add operation to reduce the hardware complexity. In [31] the reciprocal of the divisor is approximated by means of bit-inversion and a right shift, and some LSBs of the inverted signal are also truncated for further simplification.

B. Floating-Point Division

In floating-point arithmetic, a real number x is expressed in terms of sign Sx, exponent Ex, and mantissa Mx as:

$$x = (-1)^{Sx} \cdot 2^{Ex-bias} \cdot (1 + Mx) \tag{1}$$

with *bias* that depends on the selected format. The mantissa Mx is in the range [0, 1) and, as consequence, the quantity $(1 + Mx)$ lays in the interval [1, 2).

With reference to the single precision IEEE floating-point representation [33], considered in the rest of the paper, the number x is on 32 bits as shown in Fig. 1a. The MSB is the sign of x, the following $n_E = 8$ bits represent its exponent, and the remaining $n_M = 23$ LSBs are the mantissa. The *bias* is 127.

Let us consider the division $z = x/y$, where Sy, Ey, My and Sz, Ez, Mz that are signs, exponents, and mantissas of the real numbers y and z, respectively, and z is expressed as in (1). As shown in Fig. 1b, the sign Sz is computed by XOR-ing Sx and Sy, while the output exponent Ez is substantially computed by subtracting the exponents of the inputs and the *bias*.

The block mantissa divider, highlighted in red, implements:

$$P = \frac{1 + Mx}{1 + My} \tag{2}$$

Let us observe that P is in [0.5, 1) when $Mx < My$, while P is already in [1, 2) when $Mx \geq My$. Therefore, the integer part of P comprises only its MSB (named MSB_P in Fig. 1b) with $MSB_p = 0$ if P is in [0.5, 1), and $MSB_p = 1$ if P is in [1, 2).

In order to extract the output mantissa Mz, the quotient P should be in the form $(1 + Mz)$, and hence P should be normalized in the range [1, 2). Accordingly, the normalization block in Fig. 1b performs the following operations:

$$(1 + Mz) = \begin{cases} 2 \cdot P = 2 \cdot (1 + Mx)/(1 + My) & if\ MSB_P = 0 \\ P = (1 + Mx)/(1 + My) & if\ MSB_P = 1 \end{cases} \tag{3}$$

Thus, when $MSB_P = 0$ a left shift is required to move P in the range [1, 2). In this case the exponent has to be decremented by 1 in order to compensate for the mantissa shift. Overall, E_z can be computed as follows:

$$Ez = Ex - Ey - bias - \overline{MSB}_P \tag{4}$$

As shown in (3), the mantissa computation requires the division between $(1 + Mx)$ and $(1 + My)$, which is the most compute-intensive block.

III. Proposed Technique

Before describing our approach, let us call P' the approximate mantissa quotient, and let us define the Relative Error Distance (*RED*) between P and P' as

$$RED = \left| \frac{P - P'}{P} \right| \tag{5}$$

Starting from (5), we compute the Mean Relative Error Distance (*MRED*) as the average value of the *RED*.

In the following, section III-A shows the theoretical reasoning used to set up the minimization problem able to find optimal coefficients, while section III-B revisits the proposed approach with the aim to find the coefficients numerically. Then, in section III-C quantization effects are addressed to compute optimal quantized coefficients.

A. Theoretical Approach

As observable from (2), the quotient P can be seen as a function of two variables, Mx and My, which assumes values in the mantissas' plane defined as [0, 1) × [0, 1). In order to approximate the quotient, we partition the plane *Mx–My* in $nx \times ny$ regions as shown in Fig. 2, where each region, named tile in the following, has dimensions $\Delta x = 1/nx$ and $\Delta y = 1/ny$.

In the following, we introduce the indexes h, k to identify a tile of the plane, and call (Mx_h, My_k) the coordinates of its lower-left corner, with $Mx_h = (h - 1) \cdot \Delta x$ and $My_k = (k - 1) \cdot \Delta y$ (see Fig. 2).

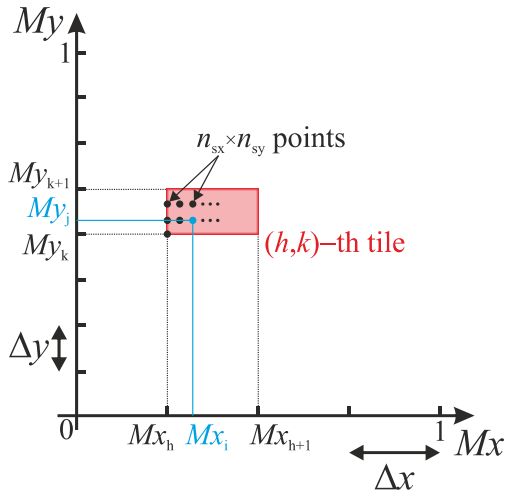


Fig. 2. Partitioning of the mantissas's plane in $nx \times ny$ tiles. In this example, nx and ny are equal to 4 and 8, respectively.

Then, in the $(h,\ k)$-th tile the mantissa quotient is approximated with a linear expression of Mx, My as in a first-order Taylor expansion:

$$P' = a_{h,k} \cdot (Mx - Mx_h) + b_{h,k} \cdot (My - My_k) + c_{h,k} \quad (6)$$

where $a_{h,k}$, $b_{h,k}$, $c_{h,k}$ are suitable constants, while Mx and My lay in the range $[Mx_{\rm h}, Mx_{\rm h+1})$ and $[My_{\rm k}, My_{\rm k+1})$, respectively.

The coefficients of the approximation are not computed as in Taylor expansion, but instead are calculated in each tile in order to minimize the *MRED* of the approximation.

To that purpose, we introduce in each tile a set of uniformly distributed $n_{sx} \times n_{\rm sy}$ sampling points $(Mx_{\rm i},\ My_{\rm j})$ whose coordinates are:

$$\begin{aligned} Mx_i &= Mx_h + i \cdot \frac{\Delta x}{n_{sx}} \quad i = 0,\ 1,\ \ldots,\ n_{sx} - 1 \\ My_j &= My_k + j \cdot \frac{\Delta y}{n_{sy}} \quad j = 0,\ 1,\ \ldots,\ n_{sy} - 1 \end{aligned} \quad (7)$$

The *MRED* is minimized for this set of sampling points. Let us consider the point $(Mx_{\rm i},\ My_{\rm j})$, highlighted in blue in Fig. 2, and let us call $P_{i,j}$ as the exact quotient: $P_{i,j} = (1 + Mx_{\rm i})/(1 + My_{\rm j})$. The relative error distance at the considered $(i,\ j)$ point is:

$$RED_{i,j} = \left| \frac{P_{i,j} - a_{h,k} \cdot (Mx_i - Mx_h) - b_{h,k} \cdot (My_j - My_k) - c_{h,k}}{P_{i,j}} \right| \quad (8)$$

The *MRED* in the $(h,\ k)$-th tile is obtained by averaging the *RED* over the set of sampling points:

$$MRED_{h,k} = \frac{1}{n_{sx} \cdot n_{sy}} \cdot \sum_{i=0}^{n_{sx}-1} \sum_{j=0}^{n_{sy}-1} RED_{i,j} \quad (9)$$

B. Minimization Problem

In order to minimize the overall *MRED* of the divider, we proceed tile-by-tile, by minimizing the $MRED_{h,k}$ of (9) for $h = 1..nx$, $k = 1..ny$. Thus, the optimization problem can be expressed as follows:

for each tile, find the coefficients $a_{\rm h,k}$, $b_{\rm h,k}$, $c_{\rm h,k}$ such that:

$$\sum_{i=0}^{n_{sx}-1} \sum_{j=0}^{n_{sy}-1} RED_{i,j} \ \text{ min!} \quad (10)$$

We can organize the problem of (10) in an equivalent form, more suitable for numerical solution. To that purpose, we introduce a set of auxiliary variables $u_{\rm i,j}$ in addition to $a_{\rm h,k}$, $b_{\rm h,k}$, $c_{\rm h,k}$, and reformulate (10) as follows:

$$\sum_{i=0}^{n_{sx}-1} \sum_{j=0}^{n_{sy}-1} u_{i,j} \ \text{ min!} \quad (11)$$

$$\text{Subject to :} \quad RED_{i,j} \le u_{i,j} \quad (12)$$

for: $i = 0..\ n_{\rm sx}-1;\ j = 0..\ n_{\rm sy}-1$

By using (8), the constraint (12) can be written as:

$$\left| P_{i,j} - a_{h,k} \cdot (Mx_i - Mx_h) - b_{h,k} \cdot (My_j - My_k) - c_{h,k} \right| \le u_{i,j} P_{i,j} \quad (13)$$

This constraint is in the form $|a| \le b$. It can be rewritten with two linear constraints not involving the absolute value:

$$|a| \le b \ \Leftrightarrow \begin{cases} a \le b \\ -a \le b \end{cases} \quad (14)$$

In this way, (13) is modified as follows:

$$\begin{aligned} -a_{h,k} \cdot (Mx_i - Mx_h) - b_{h,k} \cdot (My_j - My_k) \\ - c_{h,k} - u_{i,j} \cdot P_{i,j} \le -P_{i,j} \\ a_{h,k} \cdot (Mx_i - Mx_h) + b_{h,k} \cdot (My_j - My_k) \\ + c_{h,k} - u_{i,j} \cdot P_{i,j} \le P_{i,j} \end{aligned} \quad (15)$$

The minimization problem (11),(15) takes the form of a standard linear programming problem, defined as:

$$\begin{aligned} &\mathbf{c}^T \mathbf{x} \ \text{ min!} \\ &\textit{subject to :} \ \ \mathbf{Ax} \le \mathbf{b} \end{aligned} \quad (16)$$

where the vector x includes $3+n_{\rm sx} \cdot n_{\rm sy}$ variables corresponding to the desired coefficients $a_{\rm h,k}$, $b_{\rm h,k}$, $c_{\rm h,k}$ and the auxiliary variables $u_{\rm i,j}$. The number of constraints is $2 \cdot n_{\rm sx} \cdot n_{\rm sy}$.

In the paper the minimization problem is solved in Matlab by using the Dual Simplex method. Our numerical experiments show that having n_{sx}, n_{sy} larger than 20 does not change the solution of the optimization problem. Therefore, we fixed $n_{sx} = n_{sy} = 20$ points per tile.

The Fig. 3 highlights the behavior of the overall *MRED* of the divider for different partition schemes, defined by the parameters nx, ny. For the sake of simplicity, the values of nx, ny are chosen as powers of two. As shown in the figure, the *MRED* depends on the adopted partition scheme, exhibiting a reduction up to two orders of magnitude for increasing values of nx, ny. The error exhibits a larger dependence on ny, with best results achieved for $ny \ge 8$. For instance, considering the curve for $nx = 8$ (depicted in black), the *MRED* reduces from 10^{-2} up to about 10^{-4}. Likewise, the error steeply decreases for nx in the range [2, 16], whereas a lower reduction is

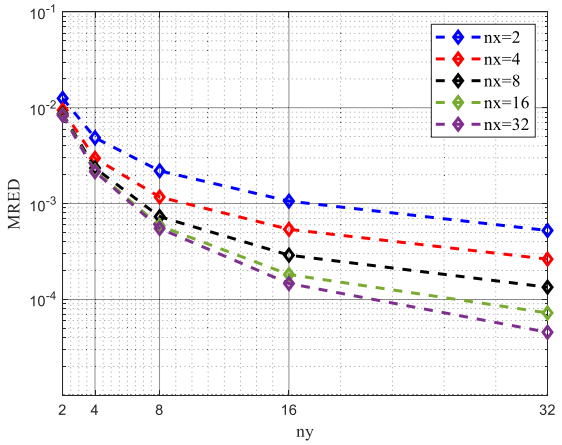


Fig. 3. Behavior of the overall MRED of the mantissa divider as function of different partition schemes.

registered when nx passes to 32 (as shown by the very close green and violet curves for $ny \leq 16$). Following the above considerations, partition schemes with nx in the range [2, 16] and ny in the range [2, 32] are suitable candidates to realize the approximate divider. At the same time, is it worth noting that the number of partitions affects the number of coefficients and, as consequence, the hardware complexity of the circuit. Therefore, a careful choice of the partition scheme is pivotal to properly balance the trade-off between accuracy of results and hardware performance.

C. Quantized Minimization Problem

An efficient hardware implementation of the proposed algorithm requires a careful quantization of the coefficients $a_{h,k}$, $b_{h,k}$, $c_{h,k}$, so that they can be efficiently stored in a look-up table. In order to find optimal solution also taking into consideration coefficient quantization, we replace $a_{h,k}$, $b_{h,k}$, $c_{h,k}$ with quantized terms $aq_{h,k}$, $bq_{h,k}$, and $cq_{h,k}$:

$$\begin{aligned} aq_{h,k} &= A_{h,k} \cdot LSB_a \\ bq_{h,k} &= B_{h,k} \cdot LSB_b \\ cq_{h,k} &= C_{h,k} \cdot LSB_c \end{aligned} \tag{17}$$

where $A_{h,k}$, $B_{h,k}$, $C_{h,k}$ are integer variables, and LSB_a, LSB_b, LSB_c are the weights of the least significant bits of the novel coefficients. Using (17) in place of $a_{h,k}$, $b_{h,k}$, $c_{h,k}$ allows to find a quantized solution to the problem (11), (15). The minimization problem takes the form of a mixed-integer linear programming problem, which searches for optimal values of the integer variables $A_{h,k}$, $B_{h,k}$, $C_{h,k}$ able to minimize the *MRED* taking as input parameters the partition scheme, defined by nx, ny, and the coefficient weights LSB_a, LSB_b, LSB_c (please note that the problem is mixed-integer since variables $u_{i,j}$ remain real). It is worth noting that, using the coefficients of (17), the mantissa quotient of (6) is computed now as:

$$P' = aq_{h,k} \cdot (Mx - Mx_h) + bq_{h,k} \cdot (My - My_k) + cq_{h,k} \tag{18}$$

The algorithm 1 details the steps needed to calculate the coefficients $aq_{h,k}$, $bq_{h,k}$, and $cq_{h,k}$.

A careful choice of the weights LSB_a, LSB_b, LSB_c allows to obtain the best trade-off between precision and hardware complexity. The lower LSB_a, LSB_b, LSB_c the better the *MRED* (due to the finer resolution of the coefficients), but the larger the hardware complexity (due to the logic to store the coefficients). Figure 4 shows the behavior of the *MRED* with quantized coefficients as function of LSB_a, LSB_b for $LSB_c = 2^{-2}, 2^{-4}, 2^{-7}, 2^{-10}$. In this analysis, we refer to the partitions $nx = 2$, $ny = 4$ (Fig. 4a), $nx = 2$, $ny = 8$ (Fig. 4b), $nx = 4$, $ny = 8$ (Fig. 4c), and $nx = 8$, $ny = 8$ (Fig. 4d), and also reports the error performance using the unquantized coefficients. The minimization problem is solved in Matlab by using the legacy branch and bound method and $n_{sx} \times n_{sy} = 20 \times 20$ points per tile. As shown, the *MRED* depends on the weights LSB_a, LSB_b, exhibiting a strong improvement for values varying from 2^{-1} to 2^{-4}. At the same time, a careful choice of LSB_c is also required. Indeed, only for $LSB_c = 2^{-7}, 2^{-10}$ the *MRED* is able to approach the value of the unquantized case, whereas the accuracy worsens for $LSB_c = 2^{-2}, 2^{-4}$.

Algorithm 1 Algorithm for Computing the Coefficients Needed to Approximate the Quotient P According to (18)

Input: The number of tiles, nx, ny; the weights of the less-significant bits used to represent in binary format the coefficients of the approximation: LSB_a, LSB_b, LSB_c.
Output (for each tile): the quantized coefficients used in the approximation: $aq_{h,k}$, $bq_{h,k}$, $cq_{h,k}$.
Steps:
1. $nsx = 20$; $nsy = 20$; // *Fixed to 20 since numerical experiments showed that larger values do not change the solution*
2. $\Delta x = 1/nx$; $\Delta y = 1/ny$;
3. for $h = 1{:}nx$
4. for $k = 1{:}ny$
5. $Mx_h = (h-1)\Delta x$; $My_k = (k-1)\Delta y$;
6. for $i = 0 : n_{sx}-1$
7. for $j = 0 : n_{sy}-1$
8. $Mx_i = Mx_h + i\,\Delta x/nsx$; $My_j = My_k + j\,\Delta\ y/nsy$;
9. compute matrix $\mathbf{A}$ and vectors $\mathbf{c}$, $\mathbf{b}$ with (11)(15)(17)
10. solve problem (16) to compute $A_{h,k}$, $B_{h,k}$, $C_{h,k}$
11. $aq_{h,k} = A_{h,k}\ LSB_a$; $bq_{h,k} = B_{h,k}\ LSB_b$; $cq_{h,k} = C_{h,k}\ LSB_c$;
12. end for;
13. end for;
14. end for;
15. end for;

IV. Hardware Implementation of the Mantissa Divider

The hardware implementation of the proposed approximate mantissa divider is depicted in Fig. 5. The values of nx and ny are powers of two. Please, note that the proposed structure will be integrated into the floating-point divider of Fig. 1 to carry out the analyses shown in the following sections.

In order to compute the quotient, each mantissa is divided into two main fields as shown in Fig. 5a, where we consider the case of an $n_M = 8$-bit mantissa and a partition scheme with $nx = 4$ for the sake of demonstration. With reference to Mx, the first field gathers 2 MSBs (highlighted in red) and constitutes the index h, used for the selection of the desired coefficients. It is worth noting that choosing nx as a power of two allows to find the value of the index just selecting $\log_2(nx)$ MSBs of the mantissa.

The second field corresponds to the mantissa without the bits of the index, and represents the difference

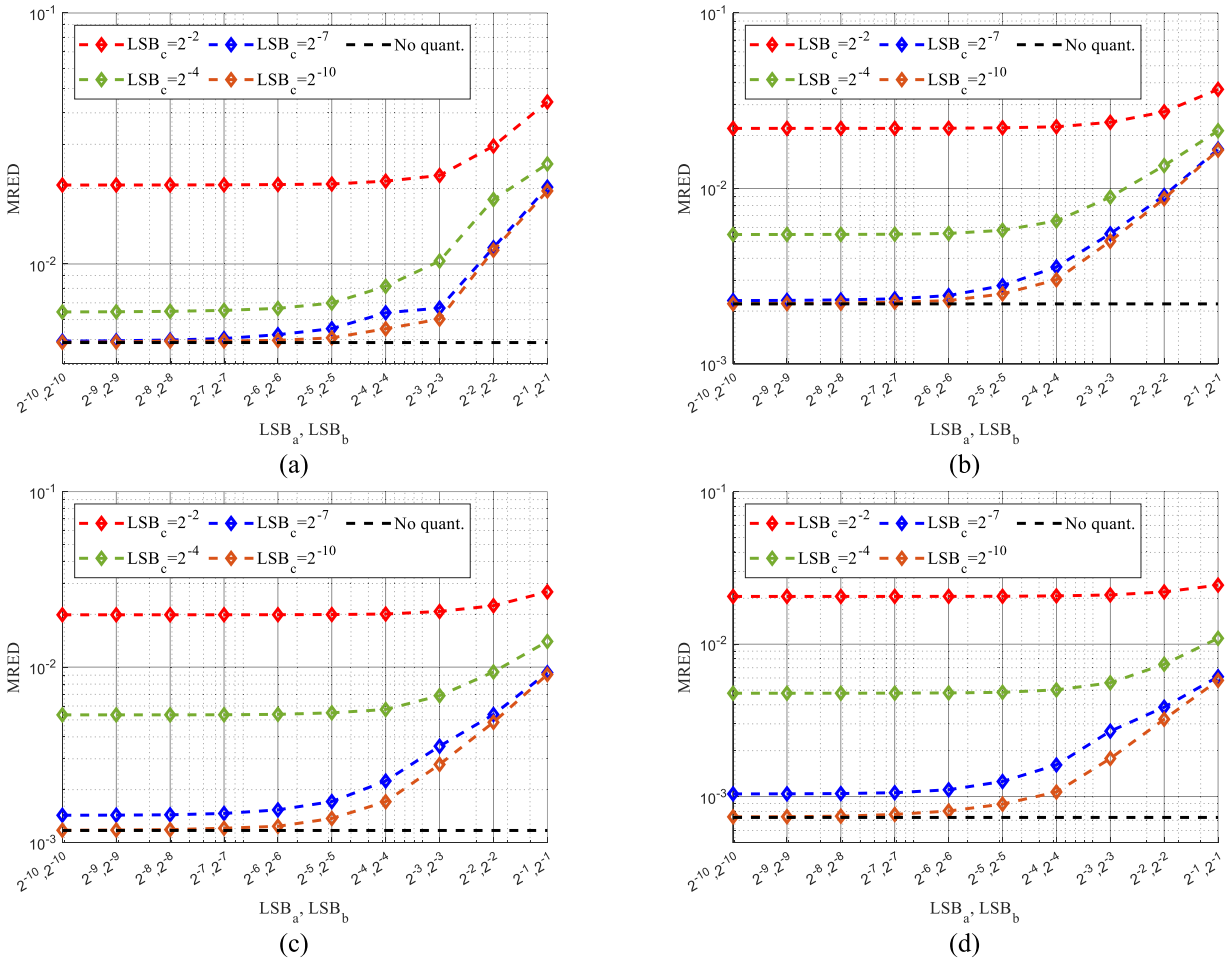


Fig. 4. *MRED* with respect to the weights $\mathrm{LSB_a}$, $\mathrm{LSB_b}$ for different values of $\mathrm{LSB_c}$ in the cases: (a) $nx = 2$, $ny = 4$, (b) $nx = 2$, $ny = 8$, (c) $nx = 4$, $ny = 8$, and (d) $nx = 8$, $ny = 8$. The dotted black line represents the error of the divider without quantization of the coefficients.

$Mx_{\mathrm{diff}} = (Mx - Mx_{\mathrm{h}})$, used to compute P' according to (18). To better clarify this point, let us consider the values of Mx in the tile $h = 3$. The lower-left corner of the tile starts with Mx_{h} =11000000, while Mx in this tile assumes the values 11000000, 11000001, …, 11111111. As consequence, the difference Mx–Mx_{h} can be obtained by simply discarding 2 MSBs of Mx. This avoids the need of a subtracter to compute Mx–Mx_{h} with beneficial effects on hardware performance.

As shown in Fig. 5(a), we discard t LSBs of Mx_{diff} (highlighted in gray in the figure), to further reduce the hardware complexity. Accordingly, Mx_{diff} is expressed on $n_{\mathrm{M}} - \log_2(nx) - t$ bits.

The mantissa My is segmented in the same way, with $\log_2(ny)$ MSBs corresponding to the index k, and the following n_{M}–$\log_2(ny)$–t bits that correspond to the truncated difference $My_{\mathrm{diff}} = (My$–$My_{\mathrm{k}})$.

The structure of the approximate mantissa divider is shown in Fig. 5b, particularized to the case of 8-bit mantissas with $t = 3$ truncated bits and partition scheme $nx = 4$, $ny = 4$. The indexes h, k allow to address the coefficients $A_{\mathrm{h,k}}$, $B_{\mathrm{h,k}}$, $C_{\mathrm{h,k}}$, stored in a LUT. The LUT will be synthesized with standard cells and stores the coefficients computed at design time. The truncated mantissas and the coefficients are multiplied and added as reported in (18). The multipliers and the adder are fused in a unique carry-save adder block (CSAB). For the sake of demonstration, Fig. 5c shows the schematic of the CSAB again in the case of mantissas on $n_{\mathrm{M}} = 8$ bits, $t = 3$ truncated LSBs. Coefficients $A_{\mathrm{h,k}}$, $B_{\mathrm{h,k}}$ are supposed to be on 2 bits with $\mathrm{LSB_a} = \mathrm{LSB_b} = 2^{-1}$, while $C_{\mathrm{h,k}}$ is on 7 bits with $\mathrm{LSB_c} = 2^{-6}$. In this example, the LSBs of the truncated mantissas have weight 2^{-5}. Therefore, the products $aq_{\mathrm{h,k}} \cdot Mx_{\mathrm{diff}}$, $bq_{\mathrm{h,k}} \cdot My_{\mathrm{diff}}$ have LSB whose weight is 2^{-6}.

The truncation allows to express the approximate quotient P' on a reduced number of bits as also shown in the example of Fig. 5c, where only 7 bits are computed instead of 9. It is worth noting that the reduced bit-width of P' allows also to simplify the multiplexer in the normalization block of the floating-point divider (visible in Fig. 1b), with further hardware improvement on the overall circuit. After normalization, the extracted mantissa is extended up to the number of bits defined by the standard (e.g. 23 bits in single precision format) by left-side zero padding.

V. Design Performance

A. Error Metrics

In order to compute the error metrics, the floating-point divider integrating the proposed mantissa divider is simulated

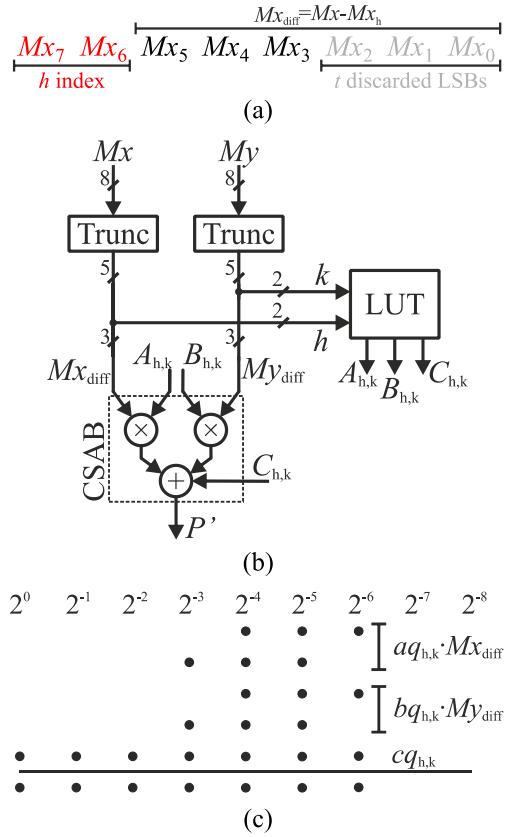


Fig. 5. (a) Partition of the mantissa Mx in the case $n_M = 8$ bits, (b) schematic of the proposed approximate mantissa divider, and (c) detail of the carry save adder block (CSAB) with $n_M = 8$ and $t = 3$ truncated LSBs. Coefficients $A_{h,k}$, $B_{h,k}$ are expressed on 2 bits with $LSB_a = LSB_b = 2^{-1}$, while $C_{h,k}$ is on 7 bits with $LSB_c = 2^{-6}$.

with 10^6 couples of random inputs. Then, the obtained results are compared to the exact case. Let us name z and z' as the exact and the approximate floating-point quotients, respectively. The relative error distance is defined as $RED = |(z - z')/z|$, whereas the Mean Relative Error Distance is $MRED = mean(RED)$, where $mean(\cdot)$ is the mean value operator. We also introduce the error bias $EB = mean((z - z')/z)$, the probability of having RED greater than 2% ($PRED$), and the normalized mean error distance referred to the mantissa divider, defined as $NMED = mean(|P - P'|)/P_{max}$, with P_{max} that is the maximum possible value of the mantissa quotient. Moreover, we assess the mean absolute error $MAE = mean(|z\text{-}z'|)$, and the root mean square error $RMSE$.

Our approach has several design parameters (the number of tiles nx, ny, the coefficient weights LSB_a, LSB_b, LSB_c, the number t of truncated bits).

The selection of these parameters involves a trade-off between precision and hardware complexity.

- Increasing nx, ny improves the accuracy (owing to the smaller tiles size) but calls for a large number of coefficients and hence in larger look-up tables to store them.
- Reducing LSB_a, LSB_b, LSB_c improves the $MRED$ (due to finer resolution of the coefficients), but increases the hardware complexity (due to the larger word size used to store the coefficients).
- Increasing the number of truncated bits, t, simplifies the hardware at the cost of precision.

Defining the combination of design parameters resulting in the optimal hardware-accuracy trade-off is not a straightforward task. Nevertheless (considering the trends described above) some trial and errors are sufficient to obtain very effective and most likely optimal implementations (reported in Table I and Fig. 9 as a function of the desired precision targets), that overcome the state-of-the-art.

We consider the following partition schemes, offering a good compromise between hardware performance and precision: (i) $nx = 2$, $ny = 4$, (ii) $nx = 2$, $ny = 8$, (iii) $nx = 4$, $ny = 8$, (iv) $nx = 8$, $ny = 8$. In addition, quantized coefficients are computed with weights $LSB_a = 2^{-2}$, $LSB_b = 2^{-2}$, and $LSB_c = 2^{-7}$. The values of the coefficients for the four partition schemes are reported in Fig. 6, grouped in the form $A_{h,k}$, $B_{h,k}$, $C_{h,k}$ for each combination of the indexes h, k.

Figure 7 depicts the behavior of the $MRED$ as a function of the number t of truncated LSBs of the mantissas. The $MRED$ is practically constant for $t \leq 16$, whereas an increase in the error is shown when more than $t = 17$ bits are truncated.

The nomenclature used for the proposed floating-point architecture is $FPD2D_{nx\times ny,t}$, which points out the implemented partition scheme and the number of truncated bits in the mantissa divider. In particular, in the following we focus our attention on the designs $FPD2D_{2\times4,19}$, $FPD2D_{2\times8,18}$, $FPD2D_{2\times8,17}$, $FPD2D_{4\times8,17}$, $FPD2D_{8\times8,17}$, $FPD2D_{8\times8,16}$, $FPD2D_{8\times8,15}$ that show a favorable tradeoff between accuracy and hardware complexity.

Table I collects the error performance of the above designs, also offering a comparison with the state-of-the-art. After a careful analysis, we select for our study the implementations of [21], [24], [26], [27], and [29], named ALD, FPAD, LPCAD(k, t), CADE(N, L), and $FPDME_{K,pt}$, respectively, since they offer the best hardware-accuracy trade-off among the works cited in section II-A. For LPCAD, k is the number of coefficients used in the approximation, and t is the number of preserved columns in the inner multiplier. For CADE, N is the number of LSBs used to identify the $2^{2\cdot N}$ regions in which a coefficient is defined to minimize the approximation error, and L is the number of bits used to quantize the coefficients. For FPDME, K is the number of stipes used in the mantissas' plane, while pt is the number of truncated bits. In addition, p least significant bits are truncated also in ALD, while in FPAD parameters L and A define the approximation depth and the number of adders involved for the shift-and-add operation.

As shown in Tab. I, the proposed technique is able to offer a wide range of values for the $MRED$ in dependence on the adopted partition scheme and mantissa truncation. For instance, $FPD2D_{2\times4,19}$ achieves an $MRED$ of 1.98%, whereas implementations $FPD2D_{8\times8,15}$ and $FPD2D_{8\times8,16}$ are able to approach $MRED$ =0.4%. Similar observation applies for the $PRED$, which reduces of about two orders of magnitude from the configuration $nx = 2$, $ny = 4$ to $nx = 8$, $ny = 8$. At the same time, the EB is very close to zero in all the cases with best results achieved for $ny = 8$. The proposed dividers perform well also in terms of $NMED$, MAE, and $RMSE$, offering a wide range of values. In this case, partition scheme with $nx = 8$, $ny = 8$ leads to best results with $NMED$ in the range $2.51 \times 10^{-3} - 4.84 \times 10^{-3}$ and MAE

TABLE I

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

	Floating-point divider	*MRED*	*EB*	*PRED* (>2%)	*NMED* (mant. divider)	*MAE*	*RMSE*	Area [μm^2]	Delay [ns]	Power [μW@ 100MHz]	PDP [fJ]	ADP [μm^2·ns]
	Exact divider	-	-	-	-	-	-	2156.1	9.180	3098.5	28444.2	19793.1
State-of-the-art	**ALD, p=8 [21]**	4.07%	4.07%	6.55×10^{-1}	2.15×10^{-2}	4.54×10^{-1}	83.4	44.1	0.508	6.2	3.1	22.4
	LPCAD(2,8) [26]	1.30%	-0.10%	2.21×10^{-1}	6.62×10^{-3}	1.82×10^{-1}	51.9	116.0	0.948	20.0	18.9	110.0
	LPCAD(3,4) [26]	1.94%	-1.73%	4.30×10^{-1}	1.01×10^{-2}	2.44×10^{-1}	51.9	79.1	0.862	15.8	13.6	83.7
	LPCAD(3,8) [26]	0.75%	0.08%	8.28×10^{-2}	3.78×10^{-3}	1.03×10^{-1}	25.8	139.4	0.988	23.2	23.0	137.7
	CADE N=3, L=8 [27]	1.30%	0.08%	2.19×10^{-1}	6.84×10^{-3}	1.36×10^{-1}	22.3	155.4	0.990	23.0	22.8	153.8
	CADE N=4, L=8 [27]	0.65%	0.02%	1.81×10^{-2}	3.54×10^{-3}	5.67×10^{-2}	5.6	293.7	0.990	37.9	37.6	290.8
	FPAD L3A2 [24]	3.05%	1.59%	5.82×10^{-1}	2.79×10^{-2}	2.53×10^{-1}	19.2	102.69	0.808	14.3	11.6	83.0
	FPAD L4A2 [24]	2.22%	-0.72%	4.44×10^{-1}	2.37×10^{-2}	2.23×10^{-1}	29.3	113.27	0.870	15.9	13.9	98.5
	FPDME$_{4,15}$ [29]	1.41%	0.01%	2.22×10^{-1}	1.11×10^{-2}	1.71×10^{-1}	37.4	113.3	0.704	17.3	12.9	79.7
	FPDME$_{4,17}$ [29]	1.48%	0.05%	2.37×10^{-1}	1.19×10^{-2}	1.76×10^{-1}	40.5	92.5	0.658	14.3	9.4	60.9
	FPDME$_{8,16}$ [29]	0.81%	0.03%	8.51×10^{-2}	7.53×10^{-3}	9.63×10^{-2}	17.7	142.3	0.856	24.5	21.0	121.8
	FPDME$_{16,16}$ [29]	0.60%	0.06%	2.81×10^{-2}	5.41×10^{-3}	7.16×10^{-2}	11.9	142.0	0.914	26.1	23.9	129.8
	FPDME$_{32,16}$ [29]	0.38%	0.06%	4.85×10^{-4}	3.83×10^{-3}	3.79×10^{-2}	5.3	167.8	0.938	30.2	28.3	157.4
Proposed architecture	**FPD2D$_{2\times4,19}$**	1.98%	-0.10%	4.20×10^{-1}	2.14×10^{-2}	1.87×10^{-1}	24.8	73.0	0.651	11.3	7.3	47.5
	FPD2D$_{2\times8,18}$	1.19%	0.02%	1.78×10^{-1}	1.20×10^{-2}	1.54×10^{-1}	27.5	90.2	0.689	14.2	9.8	62.2
	FPD2D$_{2\times8,17}$	0.96%	0.04%	1.12×10^{-1}	9.07×10^{-3}	1.19×10^{-1}	21.0	104.2	0.729	17.2	12.5	76.0
	FPD2D$_{4\times8,17}$	0.66%	-0.05%	2.18×10^{-2}	5.92×10^{-3}	6.99×10^{-2}	13.6	118.3	0.812	19.3	15.7	96.1
	FPD2D$_{8\times8,17}$	0.54%	0.04%	5.01×10^{-3}	4.84×10^{-3}	5.35×10^{-2}	6.8	135.5	0.829	20.4	17.0	112.3
	FPD2D$_{8\times8,16}$	0.42%	0.04%	1.59×10^{-3}	3.19×10^{-3}	4.32×10^{-2}	8.4	148.2	0.869	23.2	20.2	128.8
	FPD2D$_{8\times8,15}$	0.39%	0.04%	1.27×10^{-3}	2.51×10^{-3}	3.87×10^{-2}	6.5	161.4	0.920	25.8	23.8	148.5

(a)

$A_{h,k},B_{h,k},C_{h,k}$	h = 0	h = 1
k = 00	4, -4, 127	4, -6, 191
k = 01	3, -3, 104	3, -4, 155
k = 10	2, -2, 90	2, -3, 134
k = 11	2, -1, 73	2, -2, 112

(b)

$A_{h,k},B_{h,k},C_{h,k}$	h = 0	h = 1
k = 000	4, -4, 127	4, -6, 191
k = 001	3, -3, 116	3, -5, 175
k = 010	3, -3, 104	3, -4, 155
k = 011	3, -2, 92	3, -3, 138
k = 100	3, -2, 83	3, -3, 126
k = 101	2, -2, 83	2, -2, 120
k = 110	2, -1, 74	2, -2, 111
k = 111	2, -1, 69	2, -2, 104

(c)

$A_{h,k},B_{h,k},C_{h,k}$	h = 00	h = 01	h = 10	h = 11
k = 000	4, -4, 128	4, -5, 160	4, -6, 192	4, -6, 222
k = 001	3, -3, 115	3, -4, 144	3, -5, 173	3, -5, 200
k = 010	3, -3, 104	3, -3, 128	3, -4, 155	3, -4, 179
k = 011	3, -2, 92	3, -3, 116	3, -3, 139	3, -4, 163
k = 100	3, -2, 84	3, -2, 105	3, -3, 127	3, -3, 148
k = 101	2, -2, 81	2, -2, 100	2, -2, 119	2, -3, 140
k = 110	2, -1, 73	2, -2, 93	2, -2, 111	2, -2, 128
k = 111	2, -1, 69	2, -1, 85	2, -2, 103	2, -2, 120

(d)

$A_{h,k},B_{h,k},C_{h,k}$	h = 000	h = 001	h = 010	h = 011	h = 100	h = 101	h = 110	h = 111
k = 000	4, -4, 128	4, -4, 143	4, -5, 160	4, -5, 175	4, -6, 192	4, -6, 208	4, -6, 233	4, -7, 239
k = 001	3, -3, 115	3, -3, 128	3, -4, 128	3, -4, 157	3, -4, 170	3, -5, 186	3, -5, 200	3, -5, 213
k = 010	3, -2, 102	3, -3, 116	3, -3, 128	3, -3, 140	3, -4, 154	3, -4, 167	3, -4, 179	3, -4, 191
k = 011	3, -2, 93	3, -2, 104	3, -3, 117	3, -3, 128	3, -3, 139	3, -3, 150	3, -4, 163	3, -4, 175
k = 100	3, -2, 85	3, -2, 95	3, -2, 106	3, -2, 116	3, -3, 128	3, -3, 138	3, -3, 149	3, -3, 159
k = 101	2, -2, 81	2, -2, 90	2, -2, 100	2, -2, 109	2, -2, 119	2, -2, 128	2, -3, 140	2, -3, 149
k = 110	2, -1, 73	2, -1, 82	2, -2, 93	2, -2, 102	2, -2, 110	2, -2, 119	2, -2, 128	2, -2, 137
k = 111	2, -1, 69	2, -1, 76	2, -1, 85	2, -2, 95	2, -2, 103	2, -2, 112	2, -2, 120	2, -2, 128

Fig. 6. Coefficients of the divider in the cases (a) $nx = 2$, $ny = 4$, (b) $nx = 2$, $ny = 8$, (c) $nx = 4$, $ny = 8$, (D) $nx = 8$, $ny = 8$ computed with weights $\mathrm{LSB_a} = 2^{-2}$, $\mathrm{LSB_b} = 2^{-2}$, and $\mathrm{LSB_c} = 2^{-7}$. for each couple of coordinates h, k, coefficients are grouped in the form ($A_{\mathrm{h,k}}$, $B_{\mathrm{h,k}}$, $C_{\mathrm{h,k}}$).

in the range $3.87 \times 10^{-2} - 5.35 \times 10^{-2}$. Similarly, *RMSE* reduces up to 6.5 with FPD2D$_{8\times8,15}$. Among the other designs, FPDME$_{16,16}$, FPDME$_{32,16}$ [29], CADE $N = 4$, $L = 8$ [27], and LPCAD(3,8) [26] achieve performance comparable to the cases with $nx = 4$, $ny = 8$ and $nx = 8$, $ny = 8$, exhibiting *MRED* in the range 0.38%–0.75% and *PRED* between 8×10^{-2} and 5×10^{-4}. Results are comparable also in terms of *NMED* and *MAE*, ranging in $3.54 \times 10^{-3} - 5.41 \times 10^{-3}$ and $3.79 \times 10^{-2} - 1.03 \times 10^{-1}$, respectively. At the same time CADE $N = 4$, $L = 8$ [27] and FPDME$_{32,16}$ [29] achieve best values of *RMSE*. On the other hand, the other dividers exhibit worse accuracy, with *MRED* up to 4% in the case ALD [21].

B. Hardware Performance

The proposed divider and the state-of-the-art are synthesized in TSMC 28nm CMOS technology using Cadence Genus with a clock period T_{ck} of 10ns and supply voltage of 0.9V. Power consumption is computed by means of post-synthesis simulations and exploiting standard delay format (SDF) and

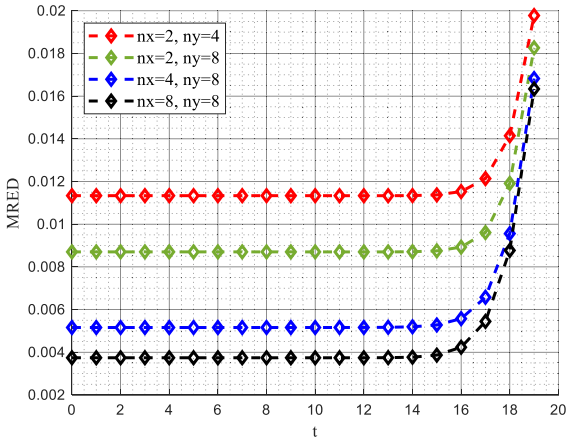


Fig. 7. *MRED* vs. the number t of truncated bits of the mantissas.

toggle count format (TCF) file to annotate net delays and the switching activity of the circuit. In addition, 10^5 random inputs are used to feed the circuits during the simulations. Area occupation and critical path delay are also investigated by means of post-synthesis analyses.

Figure 8 displays area and power breakdown in the cases of $\text{FPD2D}_{2\times4,19}$ and $\text{FPD2D}_{8\times8,15}$, offering the lowest power and the best error metrics, respectively. As shown, the mantissa divider, which gathers both the LUT and the CSAB, determines the largest part of the area and power consumption. For instance, mantissa divider of $\text{FPD2D}_{2\times4,19}$ occupies about 60% of the total area and dissipates about 54% of the overall power consumption. For $\text{FPD2D}_{8\times8,15}$ area and power of the mantissa divider are responsible for about 80% and 70% of the total area and power, respectively. Accordingly, reducing hardware complexity of mantissa divider allows to reduce the hardware complexity of the overall circuit.

Table I collects the hardware results, expressed also in terms of power-delay-product (PDP) and area-delay-product (ADP). Please note that the presented results encompass the entire floating-point divider, including mantissa division, normalization and exponent calculation.

For the comparisons, we also report hardware performance of the exact floating-point divider, chosen from the ChipAware library of the synthesizer. As shown, the proposed circuits are able to offer a remarkable improvement of performance with respect to the exact case, with PDP reducing up to a factor $\times3876$ and ADP reducing up to a factor $\times417$ in the case $\text{FPD2D}_{2\times4,19}$.

The designs $\text{FPD2D}_{2\times4,19}$ and $\text{FPD2D}_{2\times8,18}$ exhibit PDP less than 10fJ and ADP in the range from $46\mu\text{m}^2\cdot$ ns to $62\mu\text{m}^2\cdot$ ns, achieving performance close to $\text{FPDME}_{4,17}$ [29]. On the other hand, PDP increases in the case of finer partitions and less aggressive truncation. For instance, dividers with $nx = 8$, $ny = 8$ limit PDP and ADP saving up to the factors $\times1687$ and $\times176$, respectively, with results that worsen reducing the number of truncated bits. Among the other implementations, FPAD [24] and $\text{FPDME}_{4,15}$ [29] exhibit PDP reductions in the range $\times2052$–$\times2462$ and comparable values of ADP, whereas LPCAD [26] and CADE [27] show a superior complexity. At the same time, ALD [21] has good hardware performance with PDP and ADP of about 3fJ and $22\mu\text{m}^2\cdot$ ns, respectively, but it is worth noting that this performance is achieved at the cost of lower precision, as shown in the table.

To put results in perspective, Fig. 9 shows the tradeoff between PDP vs. *MRED* (Fig. 9a) and ADP vs. *MRED* (Fig. 9b). In these plots the best tradeoff is represented by the lower-left corner of the figure and the Pareto front is highlighted by the black dotted line. As shown, the proposed dividers offer the best trade-off between precision and hardware, defining the pareto front for *MRED* in the range 4×10^{-3}–2×10^{-2}.

VI. Image Application

A. JPEG Compression

JPEG compression reduces the amount of information used to represent an image, to reduce memory storage or data transmission. The algorithm initially converts the Red-Green-Blue (RGB) color space to the luminance-chrominance domain, and then utilizes cosine transformation to identify the frequency content of the selected image. Subsequently, it employs variable quantization to represent low frequencies with fine precision and high frequencies with rough accuracy. This approach enables the algorithm to primarily approximate the higher frequencies, which are less perceptible to the human eye. In addition, the algorithm also offers the possibility to define the amount of compression by acting on a quality factor Q, chosen in the range [0, 100], with $Q = 0$ and $Q = 100$ corresponding to the strongest and the finest approximation, respectively.

In our case, the proposed divider and the state-of-the-art proposals are used to implement the quantization step. For this analysis, we consider the images Autumn, Pears, Peppers, Mandrill, Light House and Fabric, from the Matlab dataset, with quality factor $Q = 35$, 70, 100. The accuracy is assessed by comparing the images compressed with exact divider with those compressed with approximate dividers. As figure of merits, we use the Mean Similarity Index (SSIM) and Peak Signal-to-Noise Ratio (PSNR), expressed in dB. Table II collects the results, showing the mean SSIM and the mean PSNR computed among the six images for each value of Q. In addition, last two columns also report the overall mean SSIM and PSNR, offering a synthetic overview of accuracy performance. As shown, proposed dividers are able to offer SSIM values very close to 1 in all the cases with average PSNR that overcomes 45dB. The accuracy increases by increasing the number of tiles, as expected. Among the different implementations, the divider $\text{FPD2D}_{8\times8,15}$ achieves the best result, with an average PSNR of 45.4dB and SSIM of 0.997. The circuits FPDME [29] are able to approach performance of the proposed divider with an average PSNR up to 44.4dB, whereas CADE $N = 4$, $L = 8$ [27] and LPCAD(3,8) [26] exhibit lower results, with PSNR around to 42dB. At the same time, implementation ALD $p = 8$ [21] offers worst accuracy, with PSNR limited to 24.3dB. Figure 10a and 10b depict the images "Pears" and "Light House" compressed with quality factor $Q = 35$ and $Q = 70$, respectively. For the sake of demonstration, we report the results obtained with the implementations ALD $p = 8$

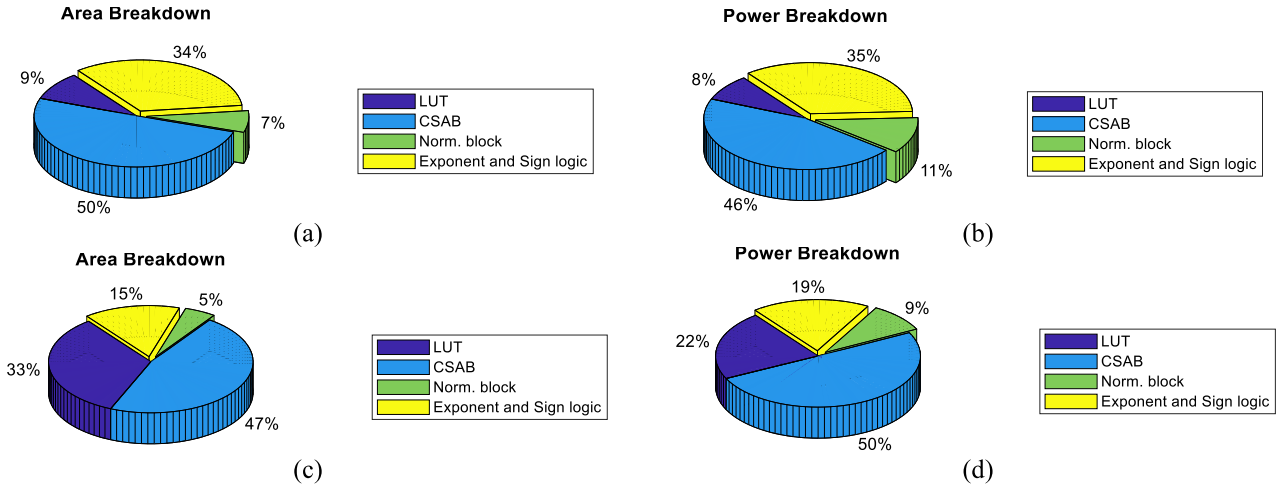


Fig. 8. (a) Area and (b) power breakdown for the divider $\text{FPD2D}_{2\times4,19}$, and (c) area and (d) power breakdown for the divider $\text{FPD2D}_{8\times8,15}$.

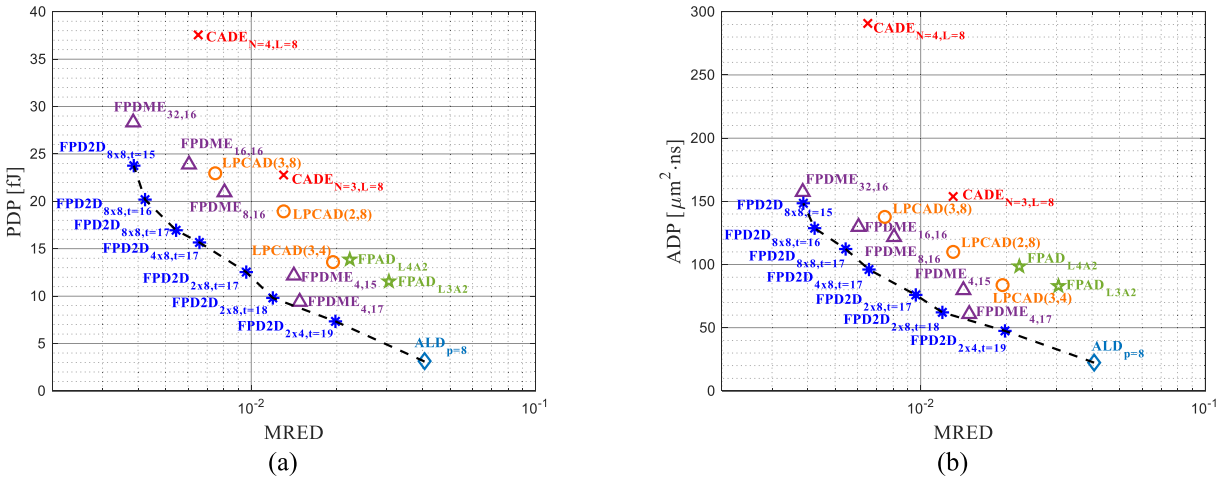


Fig. 9. Tradeoff between (a) PDP vs. *MRED* and (b) ADP vs. *MRED* for the proposed floating-point dividers and the state-of-the-art. The black dotted line represents the Pareto front.

[21], offering the worst PSNR, and $\text{FPD2D}_{8\times8,15}$, offering the best PSNR. As shown, the image obtained with the proposed divider is practically unchanged with respect to the exact case, as also confirmed by the high values of PSNR and SSIM. Conversely, ALD [21] exhibits a sensible quality degradation that is coherent with the lower values of PSNR and SSIM.

B. Tone Mapping of HDR Images

High dynamic range (HDR) images are employed in several applications such as photography, computer graphics, or medical imaging, and are produced by capturing high-quality scenes where the pixels are represented in floating-point to give a wide range of details. In these applications, tone mapping is used to convert HDR images in standard formats.

We have analyzed the performance of the proposed dividers and of the state-of-the-art to realize a tone mapping algorithm based on the Reinhard operator [32]. To this aim, the scaled luminance $L(i,\,j)$ of the HDR image is computed, where $i = 0,\,1,\,..N$, $j = 0,\,1,\,\ldots,\,M$ are the coordinates of its pixels:

$$L(i,\,j) = \frac{\alpha}{L_{w,m}} L_w(i,\,j) \tag{19}$$

where α is a parameter chosen in the range [0, 1], L_{w} is the luminance defined as function of the red, green, and blue (R,G,B, respectively) channels of the HDR image:

$$L_w(i,\,j) = 0.27\cdot R(i,\,j) + 0.67\cdot G(i,\,j) + 0.06\cdot B(i,\,j) \tag{20}$$

and $L_{\text{w,m}}$ is the geometric mean of L_{w}, defined as:

$$L_{w,m} = \exp\left(\frac{1}{NM}\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}\log(L_w(i,\,j))\right) \tag{21}$$

In the following, we consider the case $\alpha = 0.5$. As second step, the display luminance L_{d}(i,j) is computed as follows:

$$L_d(i,\,j) = \frac{L(i,\,j)}{1 + L(i,\,j)} \tag{22}$$

and is multiplied to the R,G,B channels of the HDR image:

$$\begin{aligned} R'(i,\,j) &= L_d(i,\,j)\cdot R(i,\,j) \\ G'(i,\,j) &= L_d(i,\,j)\cdot G(i,\,j) \\ B'(i,\,j) &= L_d(i,\,j)\cdot B(i,\,j) \end{aligned} \tag{23}$$

were R', G', B' are the channels with low dynamic range.

TABLE II
ACCURACY PERFORMANCE OF PROPOSED DIVIDERS AND OF THE STATE-OF-THE-ART IN JPEG COMPRESSION

	Floating-point divider	Q=35		Q=70		Q=100		Average	
		PSNR [dB]	SSIM	PSNR [dB]	SSIM	PSNR [dB]	SSIM	PSNR [dB]	SSIM
State-of-the-art	ALD, p=8 [21]	24.1	0.828	24.5	0.818	24.4	0.882	24.3	0.823
	LPCAD(2,8) [26]	32.4	0.952	38.1	0.991	44.2	0.999	38.3	0.980
	LPCAD(3,4) [26]	32.4	0.972	34.2	0.981	35.7	0.987	34.1	0.980
	LPCAD(3,8) [26]	38.8	0.992	39.2	0.991	48.0	0.999	42.0	0.994
	CADE N=3, L=8 [27]	36.9	0.986	38.0	0.989	39.8	0.991	38.2	0.989
	CADE N=4, L=8 [27]	39.6	0.993	42.4	0.996	43.8	0.997	41.9	0.995
	FPAD L3A2 [24]	29.0	0.939	39.3	0.993	29.0	0.938	32.4	0.957
	FPAD L4A2 [24]	34.7	0.982	39.7	0.994	40.9	0.995	38.4	0.990
	$\text{FPDME}_{4,15}$ [29]	27.5	0.887	35.1	0.969	44.6	0.998	35.7	0.951
	$\text{FPDME}_{4,17}$ [29]	27.8	0.897	35.2	0.971	41.8	0.996	34.9	0.955
	$\text{FPDME}_{8,16}$ [29]	39.9	0.995	35.5	0.981	42.0	0.996	39.1	0.991
	$\text{FPDME}_{16,16}$ [29]	38.9	0.993	42.2	0.997	46.6	0.999	42.5	0.996
	$\text{FPDME}_{32,16}$ [29]	41.4	0.996	41.1	0.995	50.8	0.999	44.4	0.997
Proposed architecture	$\text{FPD2D}_{2\times4,19}$	32.1	0.961	32.5	0.953	37.9	0.990	34.2	0.968
	$\text{FPD2D}_{2\times8,18}$	36.7	0.987	39.8	0.995	40.0	0.995	38.8	0.992
	$\text{FPD2D}_{2\times8,17}$	37.6	0.990	40.9	0.995	45.2	0.998	41.3	0.994
	$\text{FPD2D}_{4\times8,17}$	39.1	0.993	41.4	0.995	46.0	0.999	42.1	0.995
	$\text{FPD2D}_{8\times8,17}$	39.2	0.993	41.6	0.996	45.5	0.999	42.1	0.996
	$\text{FPD2D}_{8\times8,16}$	40.0	0.994	43.3	0.997	50.1	0.999	44.4	0.997
	$\text{FPD2D}_{8\times8,15}$	40.0	0.994	44.5	0.998	51.8	1.000	45.4	0.997

Exact compressed | ALD p=8 [21] | $\text{FPD2D}_{8\times8,15}$

(a)

Exact compressed | ALD p=8 [21] | $\text{FPD2D}_{8\times8,15}$

(b)

Fig. 10. JPEG compression results for the images (a) "Pears" with quality factor $Q = 35$ and (b) "Light House" with quality factor $Q = 70$ for the implementations ALD $p = 8$ [21] and $\text{FPD2D}_{8\times8,15}$.

As last step, the resulting channels are quantized (e.g. on 8 bits) to represent the image in a standard format. Table III collects the result of tone mapping algorithm, obtained by applying the approximate dividers for the computation of (19), (21), and (22). In this case, three test images are considered, Bottles Smal, Oxford Church, and Office, whose pixels are represented in single-precision floating-point arithmetic. As shown, the proposed dividers are able to offer high values of PSNR (up to 43.6dB on average) with SSIM very close to 1. Only the implementation CADE $N = 4$, $L = 8$ [27] is able to achieve a larger PSNR (44.8dB on average), whereas the other implementations exhibit poorer performance. Among the implementation, LPCAD [26] offers worst results, with PSNR that range between 10dB and 12dB and SSIM up to 0.658.

For the sake of demonstration, Fig. 11 reports the images obtained with LPCAD [26] and the proposed dividers in the case of the "Office" image. As shown, images computed with

TABLE III
ACCURACY PERFORMANCE OF PROPOSED DIVIDERS AND OF THE STATE-OF-THE-ART IN TONE MAPPING APPLICATION

	Floating-point divider	Bottles Small		Oxford Church		Office		Average	
		PSNR [dB]	SSIM	PSNR [dB]	SSIM	PSNR [dB]	SSIM	PSNR [dB]	SSIM
State-of-the-art	ALD, p=8 [21]	21.2	0.944	20.2	0.920	19.5	0.930	20.3	0.931
	LPCAD(2,8) [26]	14.6	0.710	10.2	0.650	11.0	0.695	11.9	0.658
	LPCAD(3,4) [26]	13.5	0.666	9.7	0.640	10.1	0.657	11.1	0.654
	LPCAD(3,8) [26]	12.3	0.633	8.4	0.590	9.1	0.623	9.9	0.616
	CADE N=3, L=8 [27]	37.7	0.997	40.6	0.993	39.2	0.995	39.2	0.995
	CADE N=4, L=8 [27]	46.0	0.999	42.6	0.997	45.7	0.999	44.8	0.998
	FPAD L3A2 [24]	28.8	0.983	31.4	0.956	34.2	0.981	31.5	0.973
	FPAD L4A2 [24]	33.0	0.993	36.3	0.974	34.0	0.986	34.4	0.984
	$FPDME_{4,15}$ [29]	37.3	0.997	36.5	0.987	31.3	0.989	35.0	0.991
	$FPDME_{4,17}$ [29]	38.3	0.997	35.6	0.986	30.7	0.989	34.9	0.991
	$FPDME_{8,16}$ [29]	39.4	0.998	41.6	0.994	32.9	0.993	38.0	0.995
	$FPDME_{16,16}$ [29]	39.7	0.998	42.7	0.995	43.7	0.998	42.0	0.997
	$FPDME_{32,16}$ [29]	49.3	1.000	38.5	0.997	41.1	0.998	42.9	0.998
Proposed architecture	$FPD2D_{2\times4,19}$	33.4	0.992	36.6	0.977	34.9	0.989	35.0	0.986
	$FPD2D_{2\times8,18}$	35.4	0.996	34.7	0.986	35.5	0.993	35.2	0.992
	$FPD2D_{2\times8,17}$	35.4	0.996	34.2	0.987	38.7	0.995	36.1	0.993
	$FPD2D_{4\times8,17}$	39.1	0.998	38.9	0.995	38.7	0.997	38.9	0.997
	$FPD2D_{8\times8,17}$	38.8	0.998	38.9	0.997	47.6	0.999	41.8	0.998
	$FPD2D_{8\times8,16}$	42.1	0.999	39.0	0.997	48.0	0.999	43.0	0.998
	$FPD2D_{8\times8,15}$	42.4	0.999	39.1	0.9997	49.2	0.999	43.6	0.998

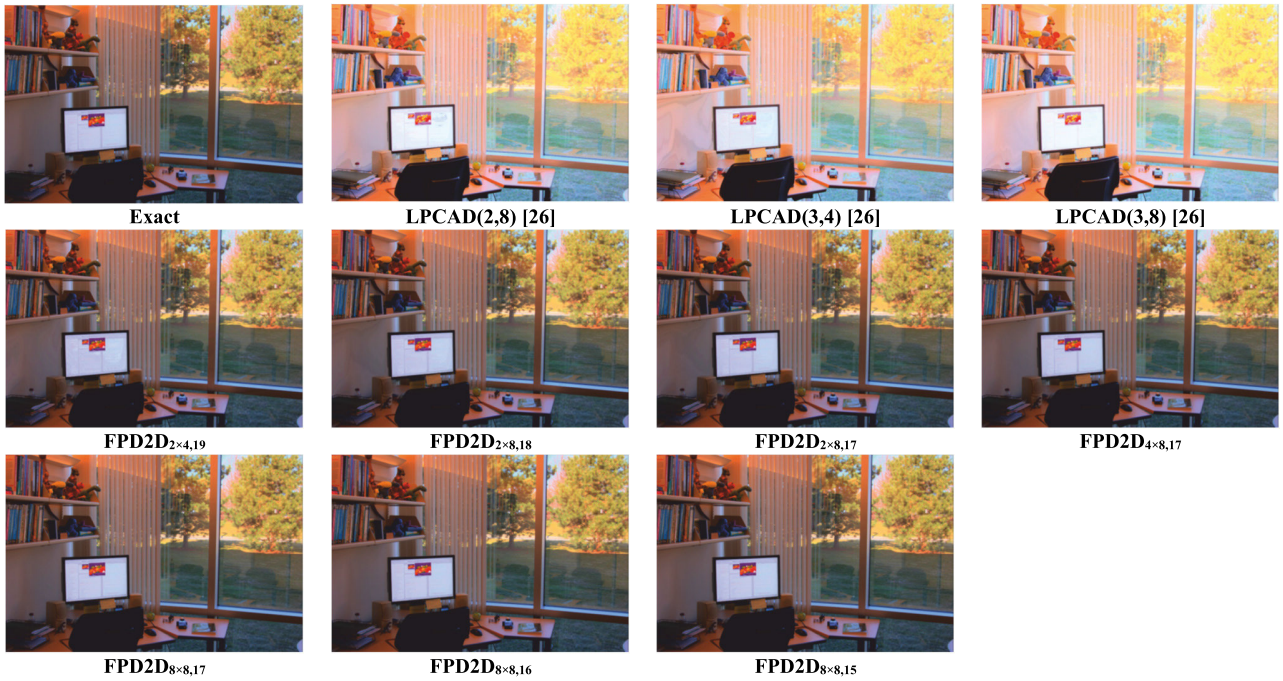


Fig. 11. Tone mapping of the "Office" image for the LPCAD implementations and the proposed dividers.

LPCAD [26] exhibit a visible alteration of the luminance level, whereas the proposed dividers allow to compute images practically unchanged with respect to the exact case.

VII. CONCLUSION

In this paper, we have proposed a novel approximate floating-point divider based on bidimensional linear approximation. In our approach, the mantissa quotient $(1 + Mx)/(1 + My)$ is revisited as function of the two variables Mx, My. The domain of this function is partitioned in $nx \times ny$ tiles, with nx, ny that are powers of two. In each tile, the quotient is approximated with a linear combination of Mx, My and a minimization problem is set up in order to find optimal quantized coefficients that minimize the *MRED* of the divider. From the hardware point of view, the proposed circuit only requires a fused multiply-and-add structure and a hardwired LUT. The proposed architecture is highly tunable at design-time over a wide range of accuracy, depending on the number of tiles chosen for the approximation. The obtained results demonstrate error performance and hardware features superior to the state-of-the-art. The proposed dividers define the Pareto front, considering the trade-off between power-delay-product vs. *MRED* and area-delay-product vs. *MRED*, for *MRED* in the range of $4 \times 10^{-3} - 2 \times 10^{-2}$.

Application results for JPEG compression and tone mapping further highlight the strength of our proposal, which exhibits Structural Similarity Index (SSIM) very close to 1 in all cases and Peak Signal-to-Noise Ratio (PSNR) up to 45 dB.

REFERENCES

[1] 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.

[2] Y.-C. Lee, T.-S. Chi, and C.-H. Yang, "A 2.17-mW acoustic DSP processor with CNN-FFT accelerators for intelligent hearing assistive devices," *IEEE J. Solid-State Circuits*, vol. 55, no. 8, pp. 2247–2258, Aug. 2020, doi: 10.1109/JSSC.2020.2987695.

[3] 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.

[4] M. Garrido, "A survey on pipelined FFT hardware architectures," *J. Signal Process. Syst.*, vol. 94, no. 11, pp. 1345–1364, Nov. 2022, doi: 10.1007/s11265-021-01655-1.

[5] Y. Guo, Z. Wang, Q. Hong, H. Luo, X. Qiu, and L. Liang, "A 60-mode high-throughput parallel-processing FFT processor for 5G/4G applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 31, no. 2, pp. 219–232, Feb. 2023, doi: 10.1109/TVLSI.2022.3227346.

[6] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proc. 50th Annu. Design Autom. Conf.*, Austin, TX, USA, May 2013, pp. 1–9, doi: 10.1145/2463209.2488873.

[7] 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.

[8] M. Imani et al., "ApproxLP: Approximate multiplication with linearization and iterative error control," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, Las Vegas, NV, USA, Jun. 2019, pp. 1–6.

[9] C. Chen, W. Qian, M. Imani, X. Yin, and C. Zhuo, "PAM: A piecewise-linearly-approximated floating-point multiplier with unbiasedness and configurability," *IEEE Trans. Comput.*, vol. 71, no. 10, pp. 2473–2486, Oct. 2022, doi: 10.1109/TC.2021.3131850.

[10] G. Di Meo, G. Saggese, A. G. M. Strollo, D. De Caro, and N. Petra, "Approximate floating-point multiplier based on static segmentation," *Electronics*, vol. 11, no. 19, p. 3005, 2022, doi: 10.3390/electronics11193005.

[11] V. Leon, T. Paparouni, E. Petrongonas, D. Soudris, and K. Pekmestzi, "Improving power of DSP and CNN hardware accelerators using approximate floating-point multipliers," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 5, pp. 1–21, Sep. 2021, doi: 10.1145/3448980.

[12] G. Saggese, E. Napoli, and A. G. M. Strollo, "CFPM: Run-time configurable floating-point multiplier," in *Proc. 18th Conf. Ph.D Res. Microelectron. Electron. (PRIME)*, Valencia, Spain, Jun. 2023, pp. 173–176, doi: 10.1109/PRIME58259.2023.10161866.

[13] M. Imani, D. Peroni, and T. Rosing, "CFPU: Configurable floating point multiplier for energy-efficient computing," in *Proc. 54th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Austin, TX, USA, Jun. 2017, pp. 1–6, doi: 10.1145/3061639.3062210.

[14] M. Imani, R. Garcia, S. Gupta, and T. Rosing, "RMAC: Runtime configurable floating point multiplier for approximate computing," in *Proc. Int. Symp. Low Power Electron. Design*, Seattle, WA, USA, Jul. 2018, pp. 1–6, doi: 10.1145/3218603.3218621.

[15] W. Liu, L. Chen, C. Wang, M. O'Neill, and F. Lombardi, "Design and analysis of inexact floating-point adders," *IEEE Trans. Comput.*, vol. 65, no. 1, pp. 308–314, Jan. 2016, doi: 10.1109/TC.2015.2417549.

[16] S. K. Beura, J. Samanta, B. P. Devi, and P. Saha, "Floating point adder using inexact mantissa adder," in *Proc. 4th Int. Conf. Comput. Commun. Syst. (I3CS)*, Shillong, India, Mar. 2023, pp. 1–5, doi: 10.1109/I3CS58314.2023.10127550.

[17] W. Liu, L. Chen, C. Wang, M. O'Neill, and F. Lombardi, "Inexact floating-point adder for dynamic image processing," in *Proc. 14th IEEE Int. Conf. Nanotechnol.*, Toronto, ON, Canada, Aug. 2014, pp. 239–243, doi: 10.1109/NANO.2014.6967953.

[18] 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.

[19] W. Liu and A. Nannarelli, "Power efficient division and square root unit," *IEEE Trans. Comput.*, vol. 61, no. 8, pp. 1059–1070, Aug. 2012, doi: 10.1109/TC.2012.82.

[20] K. J. N. S. Bhargav, S. Palisetti, and M. Rao, "A Newton Raphson method based approximate divider design for color quantization application," in *Proc. 18th Int. SoC Design Conf. (ISOCC)*, Oct. 2021, pp. 115–116, doi: 10.1109/ISOCC53507.2021.9613961.

[21] 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.

[22] 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.

[23] Z. Ebrahimi, M. Zaid, M. Wijtvliet, and A. Kumar, "RAPID: Approximate pipelined soft multipliers and dividers for high throughput and energy efficiency," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 3, pp. 712–725, Mar. 2023, doi: 10.1109/TCAD.2022.3184928.

[24] C. K. Jha, K. Prasad, V. K. Srivastava, and J. Mekie, "FPAD: A multistage 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.

[25] J. Oelund and S. Kim, "ILAFD: Accuracy-configurable floating-point divider using an approximate reciprocal and an iterative logarithmic multiplier," in *Proc. Great Lakes Symp. VLSI*, Knoxville, TN, USA, Jun. 2023, pp. 639–644, doi: 10.1145/3583781.3590262.

[26] 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.

[27] 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.

[28] 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.

[29] G. Di Meo, A. G. M. Strollo, and D. De Caro, "Novel low-power floating-point divider with linear approximation and minimum mean relative error," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 12, pp. 5275–5288, Dec. 2023, doi: 10.1109/TCSI.2023.3312974.

[30] 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.

[31] 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.

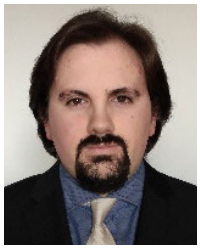
[32] Y. Kinoshita, S. Shiota, and H. Kiya, "Fast inverse tone mapping with Reinhard's global operator," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, New Orleans, LA, USA, Mar. 2017, pp. 1972–1976, doi: 10.1109/ICASSP.2017.7952501.

[33] *IEEE Standard for Floating-Point Arithmetic*, ANSI/IEEE IEEE Standard 754 2019, 2019.

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 Napoli Federico II, Italy, in 2018 and 2022, respectively. Currently, he is a Researcher with the Department of Information Technology and Electrical Engineering, University of Napoli Federico II. His research interests include the design of digital VLSI circuits for telecommunications, LMS 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 for international conferences, including PRIME, ICECS, and ESSCIRC/ESSDERC. He and his co-authors 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 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.



Luca Tegazzini received the M.S. degree (cum laude) from the University of Napoli Federico II, Italy, in 2023, where he is currently pursuing the Industrial Ph.D. degree in micro and nano electronics with the National Program. His research interests range from approximate arithmetic circuits to digitally controlled delay lines.

Ettore Napoli (Senior Member, IEEE) received the Electronic Engineering degree (Hons.) in 1995, the Ph.D. degree in electronic engineering in 1999, and the Physics degree (Hons.) in 2009. He was a Research Associate with the Engineering Department, University of Cambridge, U.K., in 2004. He was a Full Professor with the University of Napoli Federico II in 2020. He has been a Full Professor with the University of Salerno since 2021. He has authored or co-authored more than 100 papers published in international journals and conferences. His research interests include the VLSI circuit design and modeling of power semiconductor devices.

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