NOVEL LOW POWER FLOATING POINT DIVIDER WITH LINEAR APPROXIMATION AND MINIMUM MEAN RELATIVE ERROR

Rumana Tasleem[1], Dr. M. Pavithra Jyothi[2], Dr. Mohd Abdul Khader Khan[3]
[1]PG Scholar, Department of VLSI, Shadan Women's College of Engineering and Technology, Hyderabad, rumana3m@gmail.com
[2]Associate Professor, Department of ECE, Shadan Women's College of Engineering and Technology,
[3]Professor & Head, Department of ECE, Shadan Women's College of Engineering and Technology, Hyderabad,

ABSTRACT

In floating-point division, the ratio (1 + Mx)/(1 + My) is calculated, with Mx and My denoting the input values' mantissas. In this work, we provide a novel approach to approximate this process with a My-dependent linear function of Mx. In order to minimize the approximation's Mean Relative Error Distance (MRED), the coefficients are computed. In order to do this, My's range is divided into N sub-intervals, and the minimization of MRED is expressed as a linear programming problem with optimum coefficient values found in its solution. Two multipliers, an adder, and a tiny lookup table are needed for the hardware implementation. Utilizing an aggressive coefficients quantization, the design is further optimized. As N increases, obtained MRED improves, ranging from 1.4% to 0.33%. Results of implementation in a 28nm CMOS technology demonstrate that the suggested design beats the current best, providing the optimal balance between accuracy and hardware complexity. Results show excellent performance for two image processing applications: JPEG compression and change detection, with PSNR values above 50dB and SSIM values extremely near to 1.

INTRODUCTION

The design of digital signal processing (DSP) algorithms, which are widely used in everyday electronic applications, depends heavily on ARITMETIC circuits. The emergence of artificial intelligence and huge data processing necessitates the use of mathematical operations extensively for tasks like machine learning, categorization, and recognition [1]. The Internet of Things (IoT) paradigm has led to the requirement for huge amounts of data to be processed, stored, and sent. This has made the design of electrical devices with low-power characteristics difficult [2], [3].

The adoption of appropriate design solutions has become a priority in order to fulfill goal activities with acceptable power consumption since adders, multipliers, and divisions are energy-consuming circuits.

In this case, approximate computing (AC) is a useful technique that can save space and power while allowing for computation mistakes [4], [5]. Furthermore, the AC technique is very effective due to the limitations of human senses and the error-tolerant character of many real applications (e.g., image and audio processing, adaptive filtering) [6, 7, 8].

Numerous studies have focused on the design of fixed-point approximation multipliers and adders, offering numerous methods that can maximize both area and power. Papers [9], [10], and [11], for example, present a decomposition technique that splits the adder into atomic fast sub-adders, each of which processes a fraction of the input signals. Meanwhiple, papers [12], [13], and [14] make use of an approximation carry-skip architecture capable of reducing the critical path delay. The speculative approach is used to create parallel-prefix adders in [15], while approximation full-adders at the gate and transistor levels are shown in [16], [17]. Reducing the complexity of the partial product matrix (PPM) compression step usually results in significant power savings when multipliers are included. Once more, several methods have been suggested, ranging from truncation [23], [24] or input segmentation [25], [26], [27], [28], [29], to approximate compression [18], [19], [20], [21], and [22]. Appropriate correction methods are also discussed for accuracy recovery (see [20], [23], [26] for references).

In contrast to multipliers and adders, dividers have not gotten as much attention in writing. Nonetheless, hardware dividers are preferable over software implementation of the division in the design of a number of commercial microprocessors and devices [30], [31], and [32].

Iterative methods based on subtractions and multiplications are typically used in the division of two fixed-point values to compute the quotient from an initial estimate [33], [34], [35], [36], [37], [38].

Here, the design's main considerations are latency and power use. Sweeney-Robertson-Tocher (SRT) algorithms attempt to minimize the number of repetitions by utilizing redundant quotient representations and high-radix coding [38]. Additional methods to boost power include estimating the subtractor [39], using signal segmentation [40], or both [41]. An other method for computing the quotient with less energy and delay is to realize non-iterative dividers. Since it enables the division to be expressed as two-operand subtraction followed by a shift, the logarithmic number system (LNS) is a useful tool in this situation [42]. While [44] uses a linear approximation for the expression 1/y, [43] recodes the divisor y to only need a multiplication and a left-shift. While [45] proposes LNS with mean-error correction, [46] comes up with a rounding-based method to make the divider simpler. Large dynamic range and excellent precision are provided by floating-point arithmetic,

which represents integers with sign, exponent, and mantissa [47]. The design of floating-point dividers is crucial for several real-world DSP applications because of these features.

Sign and exponent calculation in a hardware divider can be easily implemented with just an XOR and a subtraction. However, the mantissa calculation is far more intricate and calls for a fixed-point division (1 + Mx)/(1 + My), where Mx and My represent the dividend and divisor mantissas, respectively. In [48], a two-step approximation method using shift and add operations is presented to do the mantissa division. In this instance, the tradeoff between hardware complexity and precision can be adjusted based on the shift and addition counts that are specified during the design phase.

A piecewise constant approximation is utilized in [49]. Similarly to [48], varying degrees of precision can be attained by appropriately selecting the number of ranges across which the constant approximation is used. In [50], a variable correction term that is kept in a LUT is used to regain accuracy after the mantissa division is estimated using subtractions. Since it affects both the accuracy and the size of the LUT, the correction term's bit count in this instance is a crucial design element. The division is reexamined in [51] as a two-variable function, and the surface of the quotient is estimated using best-fitting planes.

In this study, we suggest a novel minimal error, non-iterative approximation floating-point divider, which we will refer to as FPDME from here on. The precise operation (1 + Mx)/(1 + My) is the first step in our method, and we represent the division as a linear function of the mantissa Mx, with coefficients based on My.

The divider's accuracy is impacted by the coefficient selection. Our method finds the coefficients with the goal of minimizing the approximation's Mean Relative Error Distance (MRED). In order to do this, the range of My is divided into N sub-intervals, and the minimization of MRED is framed as a linear programming problem in each sub-interval, the solution of which yields the ideal values for the coefficients. Although we took MRED reduction into consideration, it's important to remember that our suggested strategy is easily adaptable to target other error metrics, such mean absolute error, for example.

To further improve the design, Mantissa truncation and coefficient quantization are also utilized. A lookup table (LUT) is all that is needed for hardware in the proposed division to store the coefficients. Two multipliers and an adder are combined into a single carry-save arithmetic structure. A proper selection of N and parameter quantization enables the trade-off between hardware complexity and accuracy to be adjusted during the design process.

Achieving MRED similar to or better than previously suggested approximation floating-point dividers is made possible by the proposed FPDME. In terms of power-delay product (PDP) and area-delay product (ADP), synthesis findings in TSMC 28nm CMOS technology also demonstrate an increase in hardware performances above the state-of-the-art. We showcase the outcomes of two image processing uses cases: JPEG compression and change detection. The two applications highlight the benefits of the suggested method even further, demonstrating competitive results in terms of mean structural similarity index (SSIM) and peak signal-to-noise ratio (PSNR).

LITERATURE SURVEY

Internet of Things (IoT): An overview, design components, and security concerns

The Internet of Things is a globally developing technology that facilitates the internet-based networking of sensors, automobiles, healthcare facilities, businesses, and consumers. Smart Homes, Smart Cities, Smart Agriculture, and Smart World are all made possible by this kind of construction. The vast number of devices, connection layer technologies, and services that make up the Internet of Things make its architecture very complicated. However, the most crucial factor in IoT is security. With the aid of Smart World, we provide an overview of the IoT architecture in this article. In the second section of this article, we address IoT security concerns and then IoT security solutions. Ultimately, the difficulties covered in the report may serve as avenues for future research in IoT security.

An Estimated Down-sampling Technique for Intelligent Systems with Power Limitations

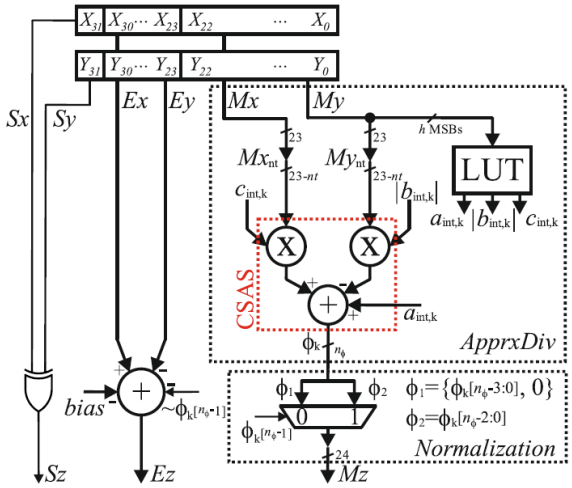
Artificial intelligence algorithms are increasingly being deployed on bespoke hardware supports in current power-constrained applications, such as the majority of Internet-of-things applications. It is imperative to minimize power consumption in various working situations, even if it means sacrificing computational precision. In order to decrease the total amount of convolution computations, we provide a unique prediction technique in this study that identifies possible dominating features in convolutional layers and then down-samples those layers. Utilizing this approximation down-sampling technique, a unique hardware architecture for Convolutional Neural Network (CNN) model inference has been designed. After using the suggested method on a number of benchmark CNN models, we were able to save up to 70% of energy overall while maintaining accuracy levels below 3% when compared to baseline designs. Experiments conducted show that the suggested architecture implemented on a Xilinx Z-7045 device and on an STM 28nm process technology dissipates only 680 and 21.9 mJ/frame, respectively, when adopted to infer the Visual Geometry Group-16 (VGG16) network model. In all scenarios, the innovative design outperforms a number of cutting-edge rivals in terms of the energy-accuracy drop product.

An emerging paradigm for energy-efficient architecture is approximate computing.
Recently, approximate computing has gained popularity as a viable method for designing digital systems that use less energy. Many systems and applications must be able to accept a certain amount of quality or optimality loss in the calculated output in order for approximate computing to work. Approximate computing approaches provide much higher energy economy by reducing the requirement for fully exact or entirely predictable computations. The design of approximation arithmetic blocks, relevant error and quality measurements, and algorithm-level approaches for approximate computing are among the latest advancements in the field that are reviewed in this work.

A New Low-Power Module-Signal Approach for the DLMS Adaptive Filter Featuring Low Steady-State Error
This work proposes a unique implementation of the Delayed LMS (DLMS) filter that can preserve regime performances while lowering power dissipation. The method is based on the fact that when the circuit is near the convergence point, the error signal has a tiny value and oscillates around zero. Consequently, there is a lot of switching activity in the feedback section's multipliers due to the error signal's most significant bits constantly flipping between positive and negative values. This research suggests using a sign-modulus representation of the error signal in order to significantly lower the filter's feedback path switching activity. Further approximation methods are also developed to minimize power dissipation even more. The suggested filter is the only one that can approach the MSE of the precise implementation with a notable reduction in power consumption, according to comparisons with the state-of-the-art. We have implemented a test-chip on TSMC 28nm CMOS technology to confirm our methodology through experimentation. According to the testing results, depending on how the filter is exactly implemented, power consumption may be reduced by up to 45.4%.

Precision-adjustable multiplier for approximative mathematical structures
In application scenarios where rigorous constraints are eased, approximation might improve performance or lower power consumption using an erroneous or simpler circuit. In applications pertaining to human senses, approximation arithmetic can yield adequate outcomes instead of than absolutely accurate results. An approximate design makes use of a trade-off between computational accuracy and power and performance. However, the level of precision needed varies depending on the application, and in certain cases, 100% precise findings are still necessary. Our research presents an accuracy-configurable approximation (ACA) adder that allows the accuracy of its outputs to be adjusted in real time. The ACA adder may function adaptively in both approximate (inaccurate) mode and accurate mode due to its configurability. Compared to traditional adder designs, the suggested adder can achieve a considerable throughput gain and total power reduction. It may be applied to applications that require accurate configuration and enhances the feasible trade-off between power/performance and quality.

PROPOSED METHODOLOGY
BLOCK DIAGRAM



Block diagram of the proposed FPDME

In this study, we suggest a novel minimal error, non-iterative approximation floating-point divider, which we will refer to as FPDME from here on. Starting with the precise operation (1 + Mx)/(1 + My), we formulate our method as a linear function of the mantissa Mx, with coefficients that depend on My.
The divider's accuracy is impacted by the coefficient selection. Our method finds the coefficients with the goal of minimizing the approximation's Mean Relative Error Distance (MRED).

MODULE EXPLANATION:
FLOATING-POINT DIVISION
The following is the representation of a real number A in floating-point notation:

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

where bias is a constant term used to shift the exponent, and S, E, and M are the sign, exponent, and mantissa of A, respectively. The bit-width of E and M as well as the bias value vary depending on the required accuracy, with one bit being used for the sign. The single precision IEEE-754 format is displayed in Fig. 1 [47]. 32 bits are needed to represent A, with unsigned values stated on 8 and 23 bits (highlighted in blue and green, respectively) for E and M. While the mantissa M fluctuates throughout the range [0, 1], the exponent E is located within [0, 255]. Furthermore, bias is adjusted to 127 to move (1)'s total exponent inside the interval [−127, 128].

Although the following assumes single precision floating-point values for the divider inputs, the suggested method is universal and works as well with other floating-point formats, such as IEEE half-precision or BFloat16. Allow us to examine the two operands in order to demonstrate the floating-point division:

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

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

The symbol for the divide Z = X/Y is comparable:

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

where values in [0, 1] are assumed for the normalization of the mantissa Mz. It's also important to remember that the number (1 + Mz) falls between [1, 2). While the modulus of Z may be expressed as follows, the sign Sz of the division is just the XOR of the operands' sign bit.

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

Now let's look at the expression (1 + Mx)/(1 + My). Its maximum value is (slightly) less than 2 when My and Mx are extremely near to zero and one, respectively. In the other scenario, a minimum value that is (slightly) more than 0.5 is attained.

Consequently, the following disparity is true:

$$0.5 < \frac{1 + Mx}{1 + My} < 2$$

It's also important to remember that when Mx > My is true, the factor (1 + Mx)/(1 + My) is greater than 1. Next, the following two scenarios are taken into consideration for the computation of Ez and Mz, beginning with (4) and (5):

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

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

In fact, when Mx > My, the quotient (1 + Mx)/(1 + My) naturally occurs in the range [1, 2] (see (6)). On the other hand, when Mx < My, (1 + Mx)/(1 + My) is in the interval [0.5, 1). As a result, the normalizing method requires that you double (1 + Mx)/(1 + My) and deduct a "1" from the exponent for compensation in order to obtain (1 + Mz) in [1, 2], as seen in (7).

Regardless, in all scenarios, the division of (1 + Mx)/(1 + My) is necessary for the mantissa computation.

PROPOSED FLOATING-POINT DIVIDER

We go over the method for approximating the division in this section. First, we represent the division as a linear function of the mantissa Mx, with coefficients that are dependent on My, dividing (1 + Mx)/(1 + My). Subsequently, we solve a minimization issue stated as a linear constraint programming problem to acquire the coefficient values that optimize the MRED. To further improve the design, we aggressively quantize the coefficients in a future phase. We recast the optimization issue as an integer linear programming problem in order to achieve this goal.

A. Approximation of Division as a Linear Function of Mx

To demonstrate the suggested method, let us first define the approximate ratio as φ(Mx, My) and the precise one as f (Mx, My) = (1 + Mx)/(1 + My). The difference between f (Mx, My) and φ(Mx, My) is the relative error distance (RED).

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

The average value of RED is represented by the MRED.

Additionally, let's rewrite the mantissa division as follows:

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

f (Mx, My) is linear with respect to Mx and has coefficients that depend on My, as shown in (9). This discovery allows us to write f (Mx, My) as follows:

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

To get the error equal to zero, we need choose g(My) = c(My) = 1/(1 + My)) from (9)–(10). However, to get the end result, c(My) has to be multiplied by Mx. Thus, it makes logical to employ two distinct approximations for g(My) and c(My), with a harsher approximation for c(My), from the standpoint of hardware implementation.

We divide the range of My into N-subintervals, each with a width of 1/N, keeping the aforementioned in mind. As seen in Fig. 2, this translates to dividing the mantissas' plane Mx − My into N horizontal stripes. Keep in mind that we select N to be a power of two in order to make it simple to identify each stripe using the most significant bits (MSBs) of My, h = log2(N). Whereas My < k/N in the k-th stripe (k − 1)/N While g(My) is estimated using a linear function of My as follows, c(My) is approximated using a constant: c(My) = ck.

g(My) = ak + bk My.

With the aforementioned presumptions, the k-th stipe's equation (10) becomes:

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

In order to estimate the quotient, this equation requires a total of 3:·N coefficients, ak, bk, and ck. Therefore, our task is to determine the coefficients that minimize the MRED.

B. The Acquisition of the Ideal Coefficients

In Fig. 2, we highlight the red dots that represent nx × ny evenly spaced locations, which we discretize to acquire the values of the coefficients ak, bk, and ck. This is where the relative error distance is calculated. Next, the relative error distance REDi,j in a generic point of coordinates (Mxi, Myj) is written as follows:

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

utilizing: j = 0, 1,... ny − 1 and i = 0, 1,... nx − 1. We might formulate our issue as follows: In order to minimize the following objective function, determine the coefficients ak, bk, and ck in each stripe:

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

It is important to note that, with the exception of a scaling factor, the summation in (13) corresponds to the MRED in the k-th stripe. As a result, reducing (13) in every stripe enables lowering the divider's overall MRED. It is important to note that, in addition to MRED, other error metrics, such as mean absolute error, might also be regarded as cost functions in equations (12) and (13).

By adding additional auxiliary variables uij, the optimization issue (13) may be further phrased as a linear programming problem so that:

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

Then, to make (13) more succinct, posing fij = f (Mxi, Myj), it may be rewritten as follows:

$$\sum_{i=0}^{nx-1} \sum_{j=0}^{ny-1} u_{ij} \text{ min!}$$

$$subject\ to:$$

$$-a_k - b_k \cdot My_j - c_k \cdot Mx_i - u_{ij} \cdot f_{ij} \le -f_{ij}$$

$$a_k + b_k \cdot My_j + c_k \cdot Mx_i - u_{ij} \cdot f_{ij} \le f_{ij}$$

$$for\ i = 0, 1, \ldots, nx - 1, \quad j = 0, 1, \ldots, ny - 1$$

where after some algebra, the restrictions are obtained from (14). The issue (15) resembles a typical linear programming problem, which looks like this:

$$\mathbf{c}^T \mathbf{x} \text{ min!}$$

$$subject\ to: \mathbf{Ax} \le \mathbf{b}$$

When there are two restrictions and the unknown vector x is made up of three + nx · ny components (ak, bk, ck, and uij for i = 0, 1,... nx − 1 and j = 0, 1,... ny − 1). The contour plot of RED is displayed in Figures 3a, 3b, and 3c for N = 4, 8, and 16, respectively. MATLAB's linprog function was used to solve the minimization issue. We will assume nx = 100 and ny = 20 in the following. Large areas of the mantissas' plane can have low RED values when N is increased, as the blue parts that grow from N = 4 to N = 16 illustrate. As a result, raising the N value also enhances the MRED. Furthermore, Fig. 3 advises appropriately selecting N to satisfy the required accuracy limitations (depending, for example, on the chosen floating-point format).

C. Coefficient Quantization

The coefficients ak, bk, and ck must have quantized values in order to implement the mantissa division in hardware. We rewrite ak, bk, and ck as follows in order to achieve this:

$$a'_k = a_{\text{int},k} \cdot LSB_a$$

$$b'_k = b_{\text{int},k} \cdot LSB_b$$

$$c'_k = c_{\text{int},k} \cdot LSB_c$$

where aint,k, bint,k, cint,k are integer variables that need to be determined, and LSBa, LSBb, and LSBc are the weights of the less-significant bits (LSB) of the coefficients (specified at design time). It is noteworthy that in order to achieve the desired precision, the selection of LSBa, LSBb, and LSBc can be appropriately adjusted based on the chosen floating-point format. We acquire a mixed-integer linear programming issue by replacing ak, bk, and ck in (15) with a′k, b′k, and c′k. This may be addressed in MATLAB by using the intlinprog tool, which returns the values of quantized coefficients that minimize the MRED.

The behavior of MRED with quantized coefficients is seen in Figure 4. As N varies from 4 to 32, the MRED in the picture is a function of LSBc, with LSBa set at 2−7 and LSBb equal to 2−1 or 2−3. Additionally, we present the inaccuracy that results from using actual, non-quantized coefficients (black dashed line). The MRED is calculated in these simulations by taking into account 106 divisions, which are carried out using 106 pairings of uniformly distributed integers stated on 23 bits. As can be seen in Fig. 4, in every case the MRED shows an impressive dependency on LSBc. As anticipated, a drop in LSBc values results in better resolutions of coefficients c′k and an increase in accuracy.

However, as Figs. 4c and 4d demonstrate, a decreased dependency on LSBb is seen, especially for N ≥16.

In actuality, the MRED obtained in this instance for LSBb = 2−3 is quite similar to that obtained for LSBb = 2−1. A suitable selection of LSBa also results in acceptable performances and is less demanding on the design. In this instance, we discovered that LSBa = 2−7 makes sense to reach a respectable MRED for small LSBc values. Finding LSBc as 2−3 for N = 4 and in the range 2−4-2−7 for N ≥ 8 yields an acceptable inaccuracy, according to the findings shown in Fig. 4. Selecting LSBb = 2−1 is thus a sensible choice. We concentrate on the following test scenarios in light of these insights in an effort to obtain reasonable hardware complexity and accurate results:

(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.
The values obtained for the coefficients aint,k, bint,k, and cint,k in the four examples under consideration are gathered in Tables I–IV.

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

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

TABLE IV

PROPOSED FLOATING-POINT DIVIDER

Fig. 5a shows the hardware implementation of the suggested FPDME. While the exponent Ez is calculated using a multi-operand adder, the sign Sz is obtained by XORing Sx and Sy. The ApprxDiv block is where the approximation mantissa division is carried out. The quantization coefficients are stored in the h MSBs of the My Index Lookup Table (LUT), and the quotient is calculated by two multipliers and an adder. Since bint,k is always negative, we save its absolute value |bint,k| in the LUT to reduce the size of the LUT. Nevertheless, as Tables I–IV demonstrate, the LUTs are quite tiny and don't require special ROM. They were synthesized with a standard-cell library in mind and specified in Verilog HDL.

By multiplying cint,k and bint,k with the mantissas and adding aint,k to the products, one may estimate the quotient φk. The signals Mxnt and Mynt are obtained by truncating the nt LSBs of mantissas in order to simplify multipliers. We emphasize that nt can be carefully selected depending on the necessary accuracy and the floating-point format being utilized. To further optimize hardware, the multipliers and adder are arranged in a fused carry-save arithmetic structure (referred to as CSAS in the picture).

COEFFICIENTS FOR N = 32, LSBA = 2^{-7}, LSBB = 2^{-1}, AND LSBC = 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

The CSAS in the example N = 8, LSBa = 2−7, LSBb = 2−1, LSBc = 2−4, and nt = 16 is depicted in detail in the figure. In this case, Mxnt and Mynt are stated on 23 − nt = 7 bits, whereas aint,k, |bint,k|, and cint,k are expressed on 8, 2, and 4 bits, respectively. Then, Mxnt · cint,k is responsible for the first four blue rows, whereas Mynt · |bint,k| is responsible for the remaining two orange rows. The word "aint,k" is shown in green. Furthermore, the products Mxnt · cint,k and Mynt · bint,k contain LSBs of weight 2−11 and 2−8, respectively, with Mxnt, Mynt having an LSB of weight 2−(23−nt) = 2−7.

It's also important to note that the CSAS computes the quotient's 12 bits rather than its full 24 bits, which enables the normalization process's hardware complexity to be reduced (explained in the

following). In general, the number of bits calculated by CSAS is nφ = 24 − nt + |log2(LSBc)|.

In order to retrieve the mantissa Mz, the Normalization block in Fig. finally rearranges φk in the interval [1, 2). As previously mentioned, the quotient fluctuates in [0.5, 2), and as a result, its MSB (shown in the picture as φk[nφ−1]) has a weight of 20. The normalizing process adds a zero at the least significant position to twice the quotient if φk[nφ−1] = 0, which places φk in the range [0.5, 1] (see the signal φ1 in the normalizing process). Additionally, ∼φk[nφ−1] is deducted to the exponent in order to account for compensation, where "∼" signifies the inversion operator.

On the other hand, no additional operation is needed if φk[nφ − 1] = 1. This indicates that φk is already in [1, 2). In this instance, Mz is represented by the fractional component of φk (refer to the signal φ2 in the picture).

A multiplexer in Fig. 5's design chooses between φ1 and φ2, and the least significant position of zeros is added to describe the outcome on 23 bits.

ASSESSMENT OF PERFORMANCES

A. Measures of Error

Let Q and Qapprx stand for the exact and approximate quotients, respectively. As demonstrated in Section II, the approximation error is defined as E = Q − Qapprx, and the relative error distance and mean relative error distance are denoted as RED = |E/Q| and MRED = avg(RED), respectively. The average operator is represented by avg(·). Additionally, we calculate the likelihood of having RED greater than 2% (referred to as PRED below) and the Error Bias, which is defined as EB = avg(E/Q) [49].

In order to calculate the error metrics, 106 divisions are made using 106 pairs of randomly distributed, single-precision floating-point values. For the purpose of accomplishing the mantissas division, we will examine examples (i), (ii), (iii), and (iv) in the following. The related floating-point dividers are designated FPDME4(7, 1, 3), FPDME8(7, 1, 4), FPDME16(7, 1, 4), and FPDME32(7, 1, 5), respectively. For reference, we also give the scenario without truncation and change the number of discarded LSBs nt. The performances of dividers [42], [44], [48], [49], and [50] are also included for comparison's purposes. The divider [42], which we will refer to as ALD from here on, processes just the first q MSB of Mx and My (q = 8 in our experiments) and subtracts mantissas in the LNS form. The work [49] uses 2d values, where d is either 2 or 3, to approach 1/(1 + My) and takes advantage of a truncated multiplier with t preserved columns. The divider [49] will be shown as LPCAD(d, t) in the following, where t = 4, 8. The mantissas' plane is divided into 2P × 2P square areas by the work [50], which will be referred to as CADE henceforth. For each section, an error compensation term represented in L bits is computed. We consider L = 8 and P = 3, 4 for our investigation. The design [44], known as TruncApp, uses just r bits to compute the quotient—r = 4 in our trials—and utilizes linear approximation for the term 1/(1 + My). Lastly, the work [48] uses two alternative shift-and-add operations (with α setting the approximation level) to realize the division. Additionally, β adders are used in each operation, and their addends are shortened on 5 bits. We refer to [48] as FPAD LαAβ in the following. The error metrics for the state-of-the-art and the suggested divider are gathered in Table V, where MRED and EB are given as percentage values. The performance of the architecture suggested in this work varies, as predicted, depending on the number of partitions N. For N = 32, the MRED increases from 1.5% to 0.33%. In addition, PRED shows a noticeable dependence, going from 2.4 × 10−1 to 3.2 × 10−4, while EB findings are nearly constant. Furthermore, nt influences the divider's accuracy; a low number of truncated LSBs results in the best approximation.

Concerning the other implementations, only LPCAD(2, 8), LPCAD(3, 8), and CADE can provide error metrics that are equivalent to the suggested FPDME; CADE, for example, can achieve an MRED of 0.65% with P = 4 and L = 8. The accuracy of the other divisions is lower, with MRED being 2% or more. The worst results are displayed in this instance by ALD and TruncApp, with MRED of around 4% and PRED of almost 7 × 10−1.

B. Hardware Performances

Using a physical flow in Cadence Genus, we synthesized the circuits in TSMC 28nm CMOS technology and detailed the suggested and cutting-edge dividers in Verilog HDL.

We have implemented FPDME4(7, 1, 3) for the proposed FPDME architecture using nt = 15 or nt = 17, whereas nt = 16 has been used for the implementation of FPDME8(7, 1, 4), FPDME16(7, 1, 4), and FPDME32(7, 1, 5). As previously indicated, the LUTs are built using the library's standard cells during the synthesis process and are defined using procedural blocks.

In the initial trial, we set a relatively lax maximum delay (10ns) on the circuits to enable the synthesizer to create least area and minimum power versions of the dividers. In this instance, we additionally generated the precise floating-point division using the synthesizer's ChipAware module.

To study the performance when a higher operating frequency is needed, we conducted a second experiment with a tighter maximum delay limitation (750 ps). Since meeting the timing limit would be impossible given the circuit's complexity, we have decided not to include the precise divider in this second experiment.

The generated netlists with 105 random inputs are simulated in both trials to determine the power

consumption. Path delays are documented in standard delay format (SDF) files, while switching activity is annotated in toggle count format (TCF) files.

The first experiment's results are included in Table VI. The power-delay product (PDP) and the area-delay product (ADP) are presented in the final two columns. Regarding the precise divider, the PDP is significantly decreased by all of the examined designs. ALD and TruncApp display the best results, with PDP in the range of 3fJ. On the other hand, these architectures also have the biggest inaccuracy.

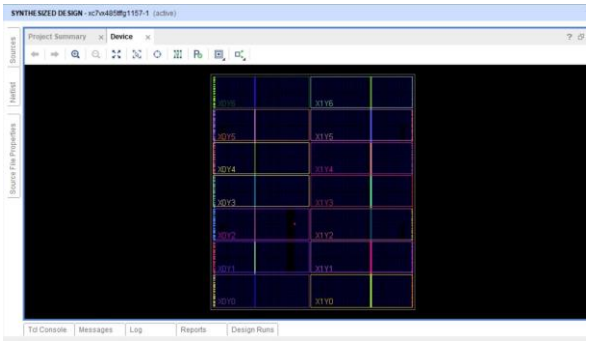
The suggested design demonstrates a reasonable balance between PDP and error. With the exception of CADE P = 4 L = 8, LPCAD(2, 8) and LPCAD(3, 8) alone, FPDME4(7, 1, 3) nt = 17 displays a lower PDP and error in comparison to all versions of LPCAD, CADE, and FPAD.

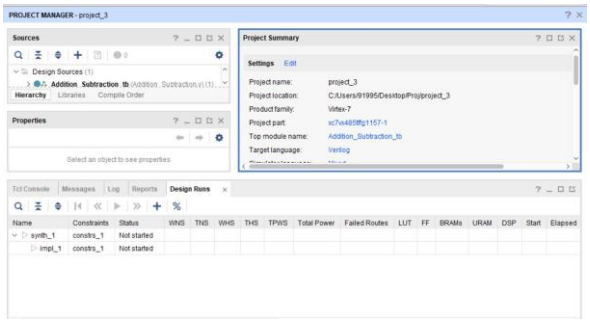
For the ADP, a similar tendency is also seen. Similarly, the findings of the second experiment are gathered in Table VII. As demonstrated, our dividers provide PDP and ADP that are on par with LPCAD, CADE P = 3, L = 8, and FPAD; FPDME4(7, 1, 3) nt = 17 yields the greatest results. Hardware complexity is best displayed by ALD and TruncApp, whereas PDP and ADP are poorer in CADE P = 4, L = 8.

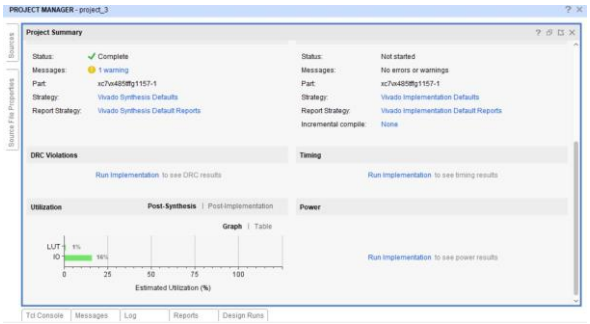
To facilitate a combined evaluation of the electrical and accuracy performances, Fig. 6 shows the PDP and the ADP for each experiment in relation to the MRED. The Pareto front is defined in this case by implementations that are closer to the bottom-left corner and have low PDP/ADP with good precision.

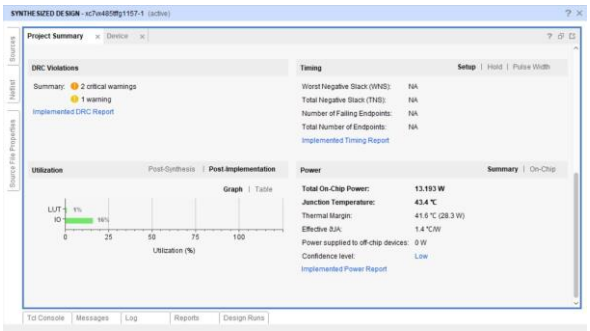
The suggested dividers, as indicated by the black dashed line in Fig. 6a, are all on the Pareto front and provide the optimal trade-off between PDP and MRED. The only implementations that behave poorly are ALD and TruncApp, with only LPCAD(3, 8) being near to the ideal curve. All other implementations, on the other hand, have a significant MRED. In order to find the optimal trade-off between ADP and MRED, the suggested FPDME are also on the pareto front. Once more, LPCAD(3, 8) yields competitive results for low accuracy, along with ALD and TruncApp.

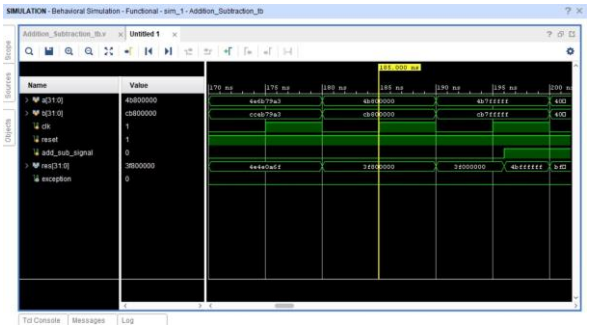
SIMULATION RESULTS:











CONCLUSION

We have presented a new non-iterative linear approximation-based approximate floating-point divider in this work. The quotient (1 + Mx)/(1 + My) has been roughly represented in our divider as a linear function of Mx with coefficients reliant on My. In order to minimize the approximation's Mean Relative Error Distance (MRED), the coefficients have been computed. In order to do this, the range of My has been divided into N sub-intervals, and the minimization of MRED has been presented as a linear programming problem in each subinterval, the solution to which

yields the ideal values for the coefficients. To further improve the design, Mantissa truncation and coefficient quantization have also been utilized.

A detailed description of the whole floating-point divider's hardware structure has been provided, and the suggested architecture's performance has been contrasted with that of earlier approximations of dividers. Based on a wide variety of mean relative error distance values, our study demonstrates that the suggested design outperforms the state of the art and provides the optimal trade-off between PDP/ADP and accuracy. Additionally, we have data for two image processing applications that demonstrate the benefits of the suggested method with competitive results in terms of Mean Structural Similarity Index (SSIM) and peak signal to noise ratio (PSNR).

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, "Dualquality 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-K7T M 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/TC. 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 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.
[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/