An Energy-Efficient, Yet Highly-Accurate, Approximate Non-Iterative Divider

Marzieh Vaeztourshizi[1], Mehdi Kamal[2], Ali Afzali-Kusha[2], Massoud Pedram[1]

[1]Ming Hsieh Department of Electrical Engineering, University of Southern California, USA

[2] School of Electrical and Computer Engineering, University of Tehran, Iran

vaeztour@usc.edu, mehdikamal@ut.ac.ir, afzali@ut.ac.ir, pedram@usc.edu

ABSTRACT

In[1] this paper, we present a highly accurate and energy efficient non-iterative divider, which uses multiplication as its main building block. In this structure, the division operation is performed by first reforming both dividend and divisor inputs, and then multiplying the rounded value of the scaled dividend by the reciprocal of the rounded value of the scaled divisor. Precisely, the interval representing the fractional value of the scaled divisor is partitioned into non-overlapping sub-intervals, and the reciprocal of the scaled divisor is then approximated with a linear function in each of these sub-intervals. The efficacy of the proposed divider structure is assessed by comparing its design parameters and accuracy with state-of-the-art, non-iterative approximate dividers as well as exact dividers in 45nm digital CMOS technology. Circuit simulation results show that the mean absolute relative error of the proposed structure for doing 1 32-bit division is less than 0.2%, while the proposed structure has significantly lower energy consumption than the exact divider. Finally, the effectiveness of the proposed divider in one image processing application is reported and discussed.

CCS CONCEPTS

• **Hardware → Logic Circuits**: Arithmetic and datapath circuits

KEYWORDS

Approximate Divider, Piecewise Linear Approximation, Low Power, High Performance.

1. INTRODUCTION

With the growth of embedded and mobile devices, there is a high demand for energy efficient and relatively high-performance circuits. Moreover, due to increasing complexity of designs, higher power dissipations and longer delays are expected. On the other hand, there are many error resilient applications such as media processing (graphics, images, *etc.*), object recognition, and data mining, which can accept a wide range of approximate results. The main reasons for error resiliency in these systems are: (i) the benefactors of these applications, *i.e.,* humans, have imperfect perceptions, (ii) there are no golden results to seek, or (iii) the input to the system itself has redundancy or noise [1].

Consequently, exactness of computations may be abandoned in favor of approximate results so as to improve the computational speed and power efficiency of digital circuits and systems that produce the said results. In this approach, which is commonly known as *approximate computing*, hardware building blocks may be simplified compared to the blocks performing exact computations, resulting in more energy efficient, higher speed, and lower area implementations [2]. An important application of approximate computing is in arithmetic blocks, which consume a large portion of the computing system power consumption [3]. Among the four basic arithmetic operations, the division operation is a less frequently-used operation in executing most applications [4]. However, when it is needed, it constitutes a costly operation because of its high latency and power consumption. Therefore, improving computational speed and reducing energy consumption of a divider block is very important in designs that perform the division operation [5].

The division operation can be performed iteratively or non-iteratively. Iterative division algorithms can themselves be categorized into two main classes: digit recurrence methods and functional iterative methods [6]. In digit recurrence algorithms, the basic operation is the subtraction operation. The bits (digits) of the quotient are obtained iteratively, one bit (digit) per step. Therefore, these dividers are easy to implement, but also become slow. The SRT division algorithm is the most frequently implemented algorithm from this class of dividers [7]. The basic operation in functional iterative algorithms is multiplication, where the algorithm starts from an estimation, and then tries to converge to the actual result. The Newton-Raphson algorithm [8] is one of the most popular functional iterative methods. On the other hand, division operation can be performed in one step by simplifying it to a multiplication operation. In this division algorithm, the dividend is multiplied by the reciprocal (inverse) of the divisor. Therefore, the complex division operation is converted into a multiplication. The challenge, of course, is to obtain the reciprocal of the divisor. One solution is the use of the approximate computing paradigm to determine an inexact value of the reciprocal of the divisor.

While many approximate designs have been suggested for adders and multipliers (*e.g.,* [9], [10]), designing approximate dividers received limited attentions (*e.g.,* [3], [11]). In this paper, we propose a highly accurate yet energy efficient approximate divider, which its computation core is based on multiplying the dividend by the reciprocal of the divisor. This structure utilizes piecewise linear approximation to calculate the reciprocal of the divisor. The interval representing the divisor is partitioned into many equal-width, non-overlapping sub-intervals, and the reciprocal is approximated with a line in each of these individual regions. In this structure, the dividend and the divisor are reformed, scaled and rounded before calculating the reciprocal and performing the final multiplication, leading to simplified hardware implementation. In this work, the final division result is obtained by employing exact multiplication, while one may use approximate multiplication as well.

The rest of the paper is organized as follows. In Section 2, a literature review on approximate dividers is provided. The algorithm of the proposed divider and its hardware implementation details are discussed in Section 3. In Section 4, the accuracy of the proposed divider is compared with some prior approximate

ISLPED '18, July 23-25, 2018, Seattle, WA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5704-3/18/07. . . $15.00
https://doi.org/10.1145/3218603.3218650

dividers, while the design parameters and the efficacy of the proposed structure in an image processing application are studied in Section 5. Finally, the paper is concluded in Section 6.

2. Related Work

One basic method to design an approximate divider is to substitute the hardware elements in the exact divider circuit with their simplified blocks. For example, in reference [12] some circuit blocks of the exact subtractor in a non-restoring divider are replaced with approximate ones to improve the speed and reduce the power consumption of the circuit. Another common technique to improve the performance of an arithmetic operation is to (dynamically) ignore the least significant bits of the operands, which is called truncation. For example, in reference [2] the dividend and the divisor are dynamically truncated to some number of bits, and therefore, a smaller exact divider is employed to perform a larger approximate division operation. There are some state-of-the-art approximate dividers, which their structure is based on multiplication of the dividend and the reciprocal of the divisor. For example, in reference [3] an approximate divider called SEERAD is presented, where the divisor ($i.e.$,B) is rounded to the nearest number in the form of $B_r = 2^{k+L}/D$, and the division operation of A/B is simplified to calculating the shifted value of $A \times D$. In this equation, k denotes the position of the leading one in B, while the tuple (L, D) is determined for each group of B values to minimize the error of the division operation. The number of groups shows the accuracy level of SEERAD. In reference [11], an approximate divider structure, called TruncApp, is suggested, where the input operands are both scaled to lie in the range of $[1, 2)$ and then truncated down to a bit width of t ($i.e.$, truncation width). Next, using Taylor series expansion, the reciprocal of the divisor is approximately calculated by inverting all the fractional bits of the truncated value of the scaled divisor and concatenating a “1” at its most significant position. As a result, the reciprocal will lie in the range of $(0.5, 1]$. Finally, the truncated scaled dividend is multiplied by the approximate reciprocal of the truncated scaled divisor to yield the approximate quotient. When the truncated lengths of the operands are increased beyond a relatively small value say 4 bits, the accuracy of the divider starts to degrade. This is due to the fact that the approximate reciprocal calculation yields to less accurate results as its bit width is increased. This is a key rationale behind our present work, which proposes a completely different method of calculating the approximate reciprocal of the divisor.

The idea of using curve fitting approaches for division was suggested in [13] without suggesting a hardware implementation for the division operation. In [14], the division operation is simplified by fitting surfaces on the quotient. To improve the accuracy, the quotient surface has been partitioned into different regions, and a square or triangular plane is fitted to each of them. This approach suffers from the need for huge look-up tables to store all the required coefficients. In our work to be presented below, to reduce the needed storage for curve fitting coefficients and also enjoy the low complexity of multiplication-based division, we suggest using piecewise linear approximation for determining the reciprocal of the divisor.

3. Proposed Approximate Divider

An n-bit unsigned integer number N may be represented as

$$N = \sum_{i=0}^{n-1} \left(x_i \times 2^i\right) = F \times 2^k = (1 + f) \times 2^k \tag{1}$$

where x_i is either zero or one, F (which will be called the *reformed* operand) is a fractional number between one and two, f is the fractional part of F, and k denotes the position of the leading one in N. Now, based on the reforming approach of (1), the division operation (A divided by B) may be represented by

$$\frac{A}{B} = \frac{F_A \times 2^{k_A}}{F_B \times 2^{k_B}} = \frac{F_A}{F_B} \times 2^{k_A - k_B} = F_A \times \left(\frac{1}{F_B}\right) \times 2^{k_A - k_B} = (1 + f_A) \times X_B \times 2^{k_A - k_B} \tag{2}$$

where X_B, which its value lies in the range of $(0.5, 1]$, is the reciprocal of the scaled divisor, F_B. In this paper, we suggest utilizing a piecewise linear curve fitting approach [13] to approximate X_B. Precisely, the $[0, 1)$ interval representing f_B may be partitioned into many equal-width, non-overlapping sub-intervals, and the reciprocal is approximated with a linear function in each sub-interval. In this case, the reciprocal is obtained from

$$\frac{1}{F_B} = \frac{1}{1 + f_B} = X_B \cong \alpha_j \times f_B + \beta_j \tag{3}$$

where j denotes the index of the sub-interval where f_B lies in. Note that for simplicity, the rounded value of f_B is multiplied by coefficient α_j. The coefficients α_j and β_j are calculated with the curve fitting and should be rounded to the nearest number based on the considered width of operations. Evidently, the accuracy of the estimated reciprocal is increased by utilizing higher number of sub-intervals, however, this will also lead to higher power dissipation and delay costs. In this paper, without loss of generality, we consider three different sub-interval counts $i.e.$, 2, 4, and 8 sub-intervals. Figure 1 shows X_B as a function of f_B as well as approximated X_B under the said sub-interval counts. As the plots show, the maximum error (2.87%) arises when f_B range is partitioned into two sub-intervals.

3.1. Hardware Implementation of the Proposed Approximate Divider

Internal structure of the hardware implementation of the proposed approximate divider is shown in Figure 2. The proposed structure is like the one suggested in [11]. However, in our structure, the inverse unit, which constitutes the core of the approximate division operation, is completely changed to rely on piecewise linear approximation method of [13]. Furthermore, a Rounding unit, instead of a Truncation unit, is employed. These modifications lead to superior accuracy of the proposed design compared to the one suggested in [11]. The detailed explanation of each block is provided in the next sub-sections. It should be mentioned that the proposed structure supports unsigned division operation. To support signed operation, before the division operation, the absolute values of the input operands should be extracted, and the division operation is performed based on the extracted absolute values, and finally, the sign of the determined quotient should be set according to the signs of the input operands.

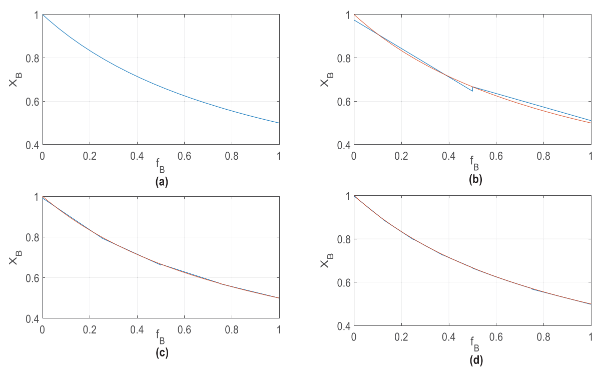

Figure 1. (a) X_B vs f_B , (b) the f_B region is partitioned into 2, (c) 4, (d) 8 sub-intervals and approximated with a straight line in each region shown with the exact curve

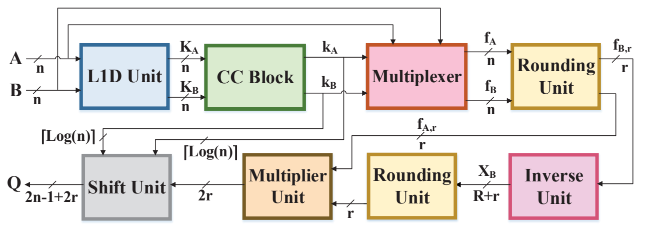

Figure 2. Hardware implementation of the proposed divider

3.1.1. Leading One Detector Unit/ Code Conversion Block/ Multiplexer

Inputs of the Leading One Detector (L1D) unit are the n-bit dividend and the n-bit divisor, which are concurrently passed to this unit. This unit has two outputs whose widths are the same as the input widths. The two outputs specify the positions of the leading one for each of the input operands. Thus, the binary representation of each output has a "1" at the position of the leading one of its corresponding input, while all other bits of the output are set to zero. The output of the L1D unit goes into a Code Conversion (CC) block, which in turn encodes the position of the leading one of the input operand in $\lceil log_2 n \rceil$ bits. The output of the CC block for each operand and the corresponding operand itself are then fed into a multiplexer, which chooses the fractional part of the reformed operand, *i.e.*, f in (1). As an example, if the dividend has an 8-bit value of $(11001110)_2$, the output of the L1D unit will be $(10000000)_2$, whereas the output of the CC block will be $(111)_2$. Finally, the output of the multiplexer will be $(10011100)_2$. Since the fractional part of a reformed 8-bit operand has at most 7 bits, we place a zero to the right side of this 7-bit fractional part to make it an 8-bit one. Similarly, if the divisor is $(01010101)_2$, the output of the L1D unit will be $(01000000)_2$, whereas the output of the CC block will be $(110)_2$. Finally, the output of the multiplexer will be $(01010100)_2$.

3.1.2. Rounding Unit

To facilitate the division operation, only a subset of most significant bits of fractional parts of both reformed operands are utilized in the division operation. Therefore, we suggest to shorten the widths of outputs of the Multiplexer unit to gain speed and power. One way to choose the most significant bits of the operands is by truncation, which was employed in [2]. However, the accuracy of rounding is more than truncation, and the accuracy of the input operand of the Inverse unit has a considerable impact on the final precision of the division operation. Therefore, in this work, we employ a Rounding unit, which rounds the operands to the nearest numbers. Outputs of this unit are designated by $f_{X,r}$, where X denotes either dividend A or divisor B and r represents the bit width of the rounded output (also called rounding width). In the previous example, the output of the Multiplexer unit with the dividend as its input was $(10011100)_2$, which means that the fractional part of the reformed operand has a value of $(0.609375)_{10}$. The output of the Rounding unit for the dividend with $r = 4$ will be $(1010)_2$, which denotes a decimal value of $(0.625)_{10}$. On the other hand, the output of the Multiplexer unit for the divisor was $(01010100)_2$, which has the decimal value of $(0.328125)_{10}$, and the output of the Rounding unit for the divisor with $r = 4$ will be $(0101)_2$ with the decimal value of $(0.3125)_{10}$. More generally, to perform the rounding operation, first, the r most significant bits of any input to the Rounding unit are truncated; next if the $(r+1)^{st}$ bit of the input is one, the truncated value is incremented by one, and this value is considered as the output of this unit. However, if $(r+1)^{st}$ bit is zero, the output will be simply the truncated value. The only exception for this operation is when the truncated r-bit value of the input of the Rounding unit is $(11 \dots 1)_2$ and its $(r+1)^{st}$ bit is one. In this case, incrementing the truncated value by one turns the rounding value to be zero, which would not be a good approximation. To avoid this, the Rounding unit will produce the truncated r-bit operand itself as its output. With respect to the above example for the dividend and the divisor, because the 5th bit of the fractional part of the reformed dividend is one, the output of the Rounding unit is $(1001)_2 + (0001)_2 = (1010)_2$. However, for the divisor, the output of the Rounding unit becomes the truncated value of $(0101)_2$, as the 5th bit is zero. The internal structure of this unit is depicted in Figure 3.

3.1.3. Inverse Unit Plus Second Rounding Unit

As mentioned before, in the proposed division algorithm, we suggest using piecewise linear approximation to estimate the reciprocal of the reformed divisor. Hence, the Inverse unit contains two one-dimensional look-up tables (LUT) with s entries each, where s is the selected number of equal-width, non-overlapping sub-intervals. LUT$_\alpha$ contains α coefficients, whereas LUT$_\beta$ contains β coefficients for the sub-intervals (see (3)). Indeed, $\lceil log_2 s \rceil$ most significant bits of $f_{B,r}$ are used as the key to these LUTs to get the stored coefficients. Since calculating $\alpha_j \times f_{B,r}$ may be costly, we use the approach suggested in [14], and thus, implement this multiplication as a series of add and shift operations. For instance, to calculate $0.625 \times f_{B,r}$, one needs to just calculate $f_{B,r}/2 + f_{B,r}/8$. Therefore, instead of storing α_j values in the corresponding table, positional notation weights of its binary representation digits are stored (*e.g.*, 2 and 8 in the case of 0.625) in r bits. To limit the required memory for weights, we round the extracted α_j to R bits. This means that value of α_j varies from 0 to $\left(1-\frac{1}{2^R}\right)$ for a width of R bits. Therefore, at most R distinct (power-of-two) weights are stored in each row of LUT$_\alpha$ using a bit vector of R bits. In this work, a maximum number of five distinct weights are sufficient to produce the required α, which, as an example, corresponds to a case when eight non-overlapping sub-intervals are selected and $r = 8$ in $f_{B,r}$. Therefore, the size of LUT$_\alpha$ is smaller than $s \times R \times r$. Since $f_{B,r}$ itself is r bits, $\alpha_j \times f_{B,r}$ will have $R + r$ bits. Thus, the bit width of β must be $R + r$, resulting in bit storage size of $s \times (R + r)$ for LUT$_\beta$. Note that we normally set $R = r$, which means the total memory needed for both LUTs is smaller than $s \times (r^2 + 2r)$.

After fetching the proper coefficients from the LUTs, a multi-operand adder is employed to find the inverse value of the input operand. More precisely, Carry Save adders (CSA) are utilized to repeatedly reduce maximum of six inputs, which are five shifted values of $f_{B,r}$ and β_j, to two outputs, and finally, a Kogge-Stone adder is utilized to sum up these levels. It is worth mentioning that X_B is always smaller than one, thus, no carry will be produced to be considered as the integer part of the result. Finally, the $2r$ bit output of the Inverse unit goes into a second Rounding unit to choose the r most significant bits required for the multiplication operation.

3.1.4. Multiplier Unit Plus Shift Unit

The last part of the division operation is to multiply the reciprocal of the rounded value of the reformed divisor to the rounded value

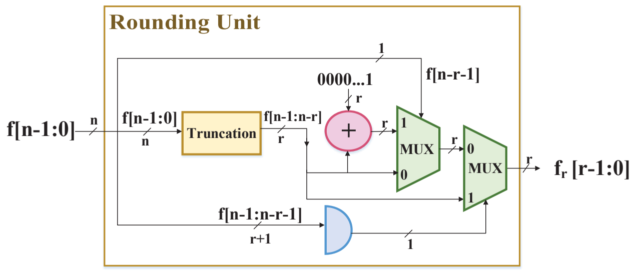

Figure 3. The internal structure of the Rounding unit

of the reformed dividend. Here, an exact Wallace Tree Multiplier is employed to perform the multiplication. Note that as mentioned before, one may use approximate multiplication to improve the performance at the cost of loss in computational accuracy. Finally, the output of the multiplier and the position of the leading ones (k_A and k_B in (2)) of the operands are passed to the Shift unit to determine the final result. The Shift unit, based on the $k_A - k_B$ value, shifts its input in a range of $(n-1)$ to $-(n-1)$ bits. Therefore, the resulting quotient (Q) is a fixed-point number with n integer bits and $n-1+2r$ fractional bits.

4. Accuracy of the Proposed Approximate Divider

In this section, the accuracy of the proposed divider is studied and compared with three state-of-the-art, non-iterative approximate dividers including SEERAD [3], TruncApp [11] and the divider proposed in [14]. The proposed divider, which we call Piecewise Linear Approximate Divider or PLApp for short, has been implemented with different number of sub-intervals (2, 4, and 8) and rounding widths (4, 5, 6, and 8); Each approximate divider is identified as PLApp(s,r), where s refers to the number of considered sub-intervals and r denotes the rounding width. In the case of SEERAD, four accuracy levels were considered, which are indicated by SEERAD(X), where X shows the accuracy level. In the case of TruncApp divider, four truncation widths have been considered, which are denoted by TruncApp(t), where t shows the considered truncation width. In the implementation of TruncApp, for a fair comparison, we employed exact multiplier similar to our structure. Finally, in the case of the divider proposed in [14], in the implemented structure, the scaled operands were rounded to the nearest number, and the plane representing the scaled quotient was partitioned into 8×8 squares. The hardware implementation of this divider was the same as the proposed divider, while the inverse unit in this structure calculated the fractional part of Q in one step. Therefore, there was no need for a Multiplier unit and the Rounding unit before it. In addition, we have implemented this divider where the input operands were integers. In the provided results, this divider is identified by Poly2D(r), where r shows the rounding width. To evaluate the accuracy of the proposed divider compared to other ones, a set of 2M uniform random inputs has been injected to all of them. The dividers have been implemented in MATLAB tool with 8, 16 and 32 bits input operands.

Figure 4 shows the mean absolute relative error (denoted by $MeanARE$) for 32-bit PLApp dividers with 2, 4 and 8 sub-intervals under different rounding widths. The absolute relative error (ARE) is defined as

$$ARE = \frac{|O_{exact} - O_{approximate}|}{O_{exact}} \quad (4)$$

where $O_{exact}(O_{approximate})$ represents the exact (approximate) value of the output. As shown in Figure 4, $MeanARE$ is improved by increasing the rounding width. Also, the divider accuracy is improved by increasing the number of non-overlapping sub-intervals. However, when the rounding width is equal to 4, the error introduced by the rounding operation outweighs the approximate reciprocal calculation. Therefore, $MeanARE$s for all the considered number of sub-intervals are almost the same. The results show the $MeanARE$ is reduced up to 77.5% (93.62%) when the number of sub-intervals (rounding widths) are increased from 2(4) to 8(8).

Table 1 shows $MeanARE$ and $VarARE$(variance of ARE) of the considered dividers. The results are sorted in descending order of $MeanARE$, where the SEERAD (Ploy2D) structure has the highest (lowest) imprecision. In all structures, except of TruncApp, increasing the width of rounding\truncation leads to higher

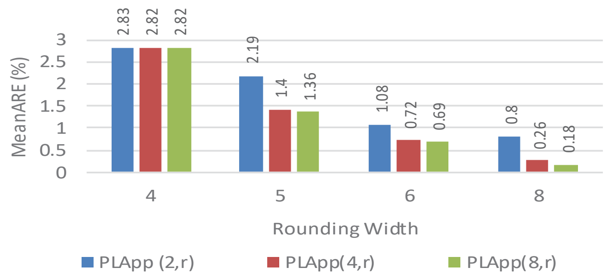

Figure 4. $MeanARE$ values for 32-bit PLApp dividers with different rounding widths

accuracy. However, in TruncApp, owing to increasing the width of input operand of the inverse unit, the imprecision of the inverse unit of this structure is increased, leading to more accuracy loss in this structure. As the results show, in all structures, the accuracy of the divider is almost independent of the input operands widths. The PLApp(8,8), which is the most accurate form of the proposed divider structure in this study, improves the $MeanARE$ and $VarARE$ by, on average, 95.66% and 99.77% compared to TruncApp(4), which is the best structure in terms of error among all considered TruncApp dividers. Also, the $MeanARE$ and $VarARE$ of PLApp(2,4), which has the lowest accuracy among all PLApp dividers, are about 30.28% and 45.88% on average lower than those of TruncApp(4) divider. On the other hand, Poly2D(8), which has the lowest error among all structures, only on average has 27.19% and 42.64% lower $MeanARE$ and $VarARE$, respectively, compared to the PLApp(8,8). The higher accuracy of the Ploy2D compared to the PLApp originates from the fact that Ploy2D divider employs planes to estimate the quotient, while in the proposed structure, the error of the Inverse unit is scaled due to the final multiplication.

Table 1. $MeanARE$ and $VarARE$ of SEERAD, TruncApp, PLApp and Poly2D structures under different input operands widths

Architecture	8-bit		16-bit		32-bit	
	$MeanARE$ (%)	$VarARE$ (%)	$MeanARE$ (%)	$VarARE$ (%)	$MeanARE$ (%)	$VarARE$ (%)
SEERAD(1)	16.55	1.10	16.25	1.00	16.25	1.00
SEERAD(2)	9.15	0.41	8.77	0.35	8.77	0.36
TruncApp(8)	7.72	0.14	7.91	0.14	7.90	0.14
TruncApp(6)	6.59	0.12	6.73	0.12	6.74	0.12
TruncApp(5)	5.36	0.09	5.46	0.10	5.46	0.10
SEERAD(3)	4.66	0.10	4.55	0.09	4.55	0.09
TruncApp(4)	4.1	0.08	4.07	0.07	4.07	0.07
PLApp(2,4)	2.87	0.04	2.83	0.04	2.83	0.04
PLApp(4,4)	2.87	0.04	2.82	0.04	2.82	0.04
PLApp(8,4)	2.87	0.04	2.82	0.04	2.82	0.04
SEERAD(4)	2.42	0.03	2.2	0.02	2.2	0.02
PLApp(2,5)	2.18	0.03	2.19	0.03	2.19	0.03
Poly2D(4)	1.86	0.02	1.94	0.02	1.94	0.02
PLApp(4,5)	1.42	0.01	1.40	0.01	1.40	0.01
PLApp(8,5)	1.37	0.01	1.36	0.01	1.36	0.01
PLApp(2,6)	1.11	0.01	1.08	0.01	1.08	0.01
Poly2D(5)	0.96	0.01	1.05	0.01	1.05	0.01
PLApp(2,8)	0.84	0.00	0.80	0.00	0.80	0.00
PLApp(4,6)	0.73	0.00	0.72	0.00	0.72	0.00
PLApp(8,6)	0.68	0.00	0.69	0.00	0.69	0.00
Poly2D(6)	0.40	0.00	0.48	0.00	0.48	0.00
PLApp(4,8)	0.26	0.00	0.26	0.00	0.26	0.00
Poly2D(8)	0.11	0.00	0.14	0.00	0.14	0.00

Finally, Figure 5 shows the relative error distribution of the 32-bit Poly2D(8), PLApp(8,8) and TruncApp(4), which are the most accurate dividers in their own class in this study. As the results show, the density of the outputs with ARE close to zero, in the case of the Ploy2D is more than the other structures. The density of the PLApp outputs with ARE smaller than 0.001 is about 27.14% (23X) smaller (higher) than those of the Ploy2D (TruncApp).

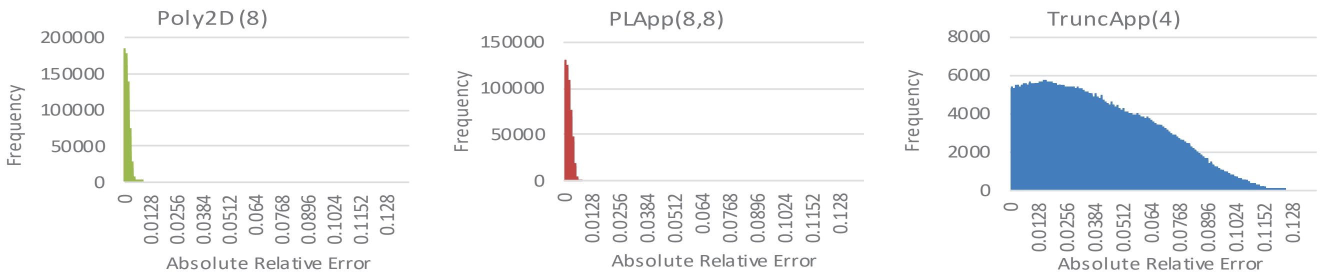

Figure 5. Output *ARE* distributions of Poly2D(8), PLApp(8,8) and TruncApp(4) structures

5. Results and Discussion

In this section, first, the design parameters of the 32-bit proposed divider are extracted and compared with the state-of-the-art approximate dividers used in the study explained in the previous section, and two exact SRT dividers. Then the efficacy of the proposed divider in one image processing application is assessed.

5.1. Design Parameters Evaluation

The dividers have been described in Verilog HDL and then synthesized using Synopsys Design Compiler in NanGate 45nm CMOS Technology [15]. The delay, power and area of the dividers have been extracted based on the Synopsys Design Compiler reports. These parameters along with energy, ED (energy-delay-product) and PDA (power-delay-area product) parameters of the studied dividers are reported in Table 2. As the results show, the SEERAD had the lowest delays among the studied structures, however, its power consumption by increasing the accuracy level considerably has been increased, and it was almost larger than that of the other approximate structures.

The TruncApp structure was more power\energy efficient than the other structures. In similar truncation\rounding widths, on average, the TruncApp structure consumed 19.34% and 38.48% lower power and energy compared to those of the PLApp structure. In addition, the ED and PDA of the PLApp structure were, on average, about 1.3X and 82.45% larger than those of the TruncApp structure.

On the other hand, all the considered design parameters of the PLApp were better than those of the Ploy2D structure. The delay, power, area, energy, ED and PDA of the proposed approximate divider were almost, on average, 14.81%, 40.78%, 33.98%, 44.89%, 47.81%, and 63.98% lower than those of the Ploy2D structure. In addition, the delay, power, area, energy, ED, and PDA of the proposed approximate divider compared to the considered exact SRT Radix-4 divider were, on average, 89.63%, 97.07%, 89.93%, 99.67%, 99.96% and 99.96% lower, respectively.

Therefore, this study shows that the design parameters of the proposed divider were better than those of the Poly2D and SEERAD (except the delay), while they were worse than those of the TruncApp. Hence, to more accurately rank these dividers, we suggest a figure of merit (FoM) to combine the effect of accuracy and design parameters in a metric. This FoM is defined by

$$FoM = Energy \times Area \times MeanARE \qquad (5)$$

Figure 6 depicts the FoMs of the 32-bit PLApp, Poly2D and TruncApp dividers under different truncation\rounding widths. By increasing the truncation\rounding widths, the efficacy of the PLApp and Poly2D were improved considerably compared to the TruncApp, which is due to the inaccuracy increase of the TruncApp structure. On the other hand, the FoMs of the PLApp, except in the case of two non-overlapped sub-intervals, were smaller than those of the poly2D. Among the PLApp divider structures, when four non-overlapped sub-intervals were employed to estimate the reciprocal led to smaller FoM.

Table 2. Delay, power, area, energy, ED, and PDA of PLApp, Poly2D, TruncApp, SEERAD, and 2 exact SRT 32-bit dividers

Architecture	Delay (ns)	Power (mW)	Area (μm^2)	Energy (pJ)	ED (pJ×ns)	PDA (pJ×μm^2)
PLApp(2,4)	1.27	0.89	1994	1.13	1.44	2253
PLApp(4,4)	1.25	0.86	1829	1.08	1.35	1975
PLApp(8,4)	1.37	0.83	1833	1.14	1.56	2090
PLApp(2,5)	1.65	1.27	2346	2.10	3.47	4927
PLApp(4,5)	1.64	1.15	2073	1.89	3.10	3918
PLApp(8,5)	1.62	1.24	2280	2.01	3.26	4583
PLApp(2,6)	1.96	1.67	2552	3.27	6.41	8345
PLApp(4,6)	1.81	1.51	2383	2.73	4.94	6506
PLApp(8,6)	2.05	1.67	2525	3.42	7.01	8636
PLApp(2,8)	2.51	2.51	3152	6.30	15.81	19858
PLApp(4,8)	2.29	2.67	3492	6.11	13.99	21336
PLApp(8,8)	2.52	2.86	3822	7.21	18.17	27557
Poly2D(4)	1.72	1.61	2638	2.77	4.76	7307
Poly2D(5)	1.84	1.99	3150	3.66	6.73	11529
Poly2D(6)	2.00	2.75	3960	5.50	11.00	21780
Poly2D(8)	2.26	4.23	5922	9.56	21.61	56614
TruncApp(4)	1.12	0.76	1738	0.85	0.95	1477
TruncApp(5)	1.30	0.97	2026	1.26	1.64	2553
TruncApp(6)	1.38	1.34	2457	1.85	2.55	4545
TruncApp(8)	1.61	1.91	3105	3.08	4.96	9563
SEERAD(1)	0.61	0.64	1343	0.39	0.24	524
SEERAD(2)	0.70	1.18	2445	0.83	0.58	2020
SEERAD(3)	0.8	2.12	4378	1.69	1.35	7415
SEERAD(4)	1.07	7.53	12451	8.06	8.62	100319
Radix-2 SRT	19.61	60.88	28691	1193	23411	34252945
Radix-4 SRT	17.63	54.47	25051	960	16930	24056628

Finally, Table 3 shows the breakdown of delay and power consumption of the proposed structure with four non-overlapped sub-intervals. As the figures of the table show, the Inverse unit (Shift unit) has almost the most impact on the delay (power consumption) of the circuit. On the other hand, the Shift Unit (L1D unit plus CC block) has almost the least influence on the delay (power consumption).

Table 3. Breakdown of the delay and power consumption of the 32-bit PLApp divider (4 sub-intervals) under different rounding widths

	Architecture	L1D+CC (%)	MUX+Round (%)	Inv (%)	Mult (%)	Shift (%)
Delay	PLApp(4,4)	15.29	19.61	30.59	23.92	10.59
	PLApp(4,5)	14.87	20.08	31.97	22.30	11.90
	PLApp(4,6)	14.52	19.03	32.58	28.71	5.16
	PLApp(4,8)	10.51	21.02	34.53	21.32	12.61
Power	PLApp(4.4)	10.78	12.70	18.48	20.19	35.58
	PLApp(4,5)	10.43	10.64	18.33	25.48	33.92
	PLApp(4,6)	8.43	10.63	16.47	29.73	33.68
	PLApp(4,8)	4.02	9.31	19.58	34.01	32.31

5.2. Image Processing Application

In this sub-section, the efficacy of the proposed divider in image division application, which detects differences in a sequence of images, has been evaluated. Therefore, in this study, each pixel of the output image was computed by dividing corresponding pixels of two consecutive images of the considered benchmarks [16]. The study has been performed on sequences benchmarks from [17], which are Walter Cronkite, Chemical Plant (close and far views),

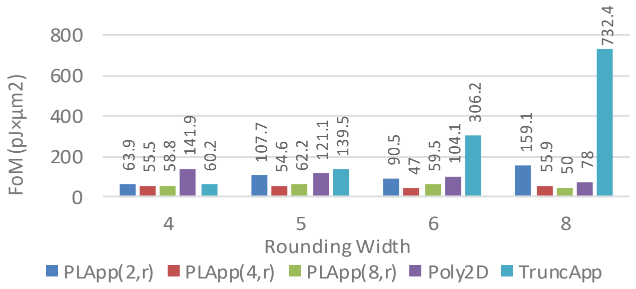

Figure 6. FoM of 32-bit PLApp, Poly2D and TruncApp dividers for different rounding widths

and Toy Vehicle benchmarks. This study has been performed in MATLAB tool by employing the models of the approximate dividers. The mean of peak signal-to-noise ratio ($MPSNR$) and mean structural similarity ($MSSIM$) [18] of the approximated results, which are obtained from comparing the result of exact division versus approximate division, are reported in Table 4 for the 16-bit TruncApp, PLApp (with four sub-intervals) and Poly2D dividers under different truncation\rounding widths. The considered architectures are sorted based on $MeanARE$ values (from highest to lowest) in this table. Note that in this study, the SEERAD was not included due to its high $MeanARE$.

As expected, the output quality of the TruncApp divider was lower than the other dividers. On average, the $MPSNR$ and $MSSIM$ of the output images of the proposed divider were about 46.79% and 2.01% larger than those of the TruncApp divider, respectively. In the similar number of rounding widths, the $MPSNRs$ of the output images of the Poly2D was about 8.94% higher than the those of the PLApp divider. However, especially in the large number of rounding widths, the $MSSIMs$ of the output images were almost similar in the cases of the PLApp and Poly2D dividers.

Table 4. $MPSNRs$ and $MSSIMs$ of the output images of image division operation realized by 16-bit TruncApp, PLApp (in the case of 4 sub-intervals) and Poly2D dividers under different benchmarks

Architecture	Walter Cronkite		Chemical Plant (close view)		Chemical Plant (far view)		Toy Vehicle	
	$MPSNR$	$MSSIM$	$MPSNR$	$MSSIM$	$MPSNR$	$MSSIM$	$MPSNR$	$MSSIM$
TruncApp(8)	29.7	0.96	29.6	0.97	28.8	0.98	27.6	0.99
TruncApp(6)	31.0	0.96	30.9	0.97	30.1	0.98	28.6	0.98
TruncApp(5)	32.7	0.95	32.5	0.97	31.6	0.98	30.0	0.97
TruncApp(4)	34.7	0.91	34.2	0.95	33.6	0.98	34.4	0.94
PLApp(4,4)	37.4	0.92	36.9	0.97	36.4	0.98	38.1	0.94
Poly2D(4)	42.1	0.96	40.1	0.98	39.2	0.99	44.7	0.98
PLApp(4,5)	43.4	0.98	43.0	0.99	41.8	1.00	43.6	0.98
Poly2D(5)	48.4	0.99	46.0	1.00	44.0	1.00	49.1	0.99
PLApp(4,6)	48.9	1.00	48.6	1.00	46.1	1.00	48.1	0.99
Poly2D(6)	54.7	1.00	52.7	1.00	49.0	1.00	53.7	1.00
PLApp(4,8)	54.4	1.00	54.0	1.00	49.7	1.00	54.5	1.00
Poly2D(8)	58.7	1.00	57.9	1.00	51.9	1.00	55.9	1.00

6. CONCLUSIONS

In this work, a low error approximate divider, which employed multiplication as its main operation, was proposed. In this structure, the input operands, first, were reformed (their representation) and scaled, and then, rounded. Next, the inverse of the rounded value of the scaled divisor was estimated by exploiting a linear piecewise approximation approach. In this approach, the interval representing the fractional part of the scaled divisor was portioned into some equal-width non-overlapping sub-intervals. Afterward, the approximated reciprocal of the rounded value of the scaled divisor and the rounded value of the scaled dividend were multiplied, and finally, the multiplication result was shifted to obtain the approximate result of the division operation. The efficacy of the proposed structure was evaluated for different numbers of sub-intervals and rounding widths. The evaluation showed that the accuracy of the proposed divider measured in terms of mean absolute relative error ($MeanARE$) was in the range of 2.87% to 0.17%, while its defined figure of merit, that combined the effect of accuracy and design parameters, was smaller than those of the state-of-the art dividers.

ACKNOWLEDGMENT

This research is supported in part by a grant from the Software Hardware Foundations of the National Science Foundation.

REFERENCES

[1] J. Han an and M.Orshansky, “Approximate computing: An emerging paradigm for energy-efficient design,” in European Test Symposium, pages 1-6, 2013

[2] S. Hashemi, R. I. Bahar, S. Reda, “A low-power dynamic divider for approximate applications,” Proceedings of the 53th ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, TX, USA, 2016, pp. 2-6

[3] 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,” Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016, pp. 1481-1484

[4] S.Amanollahi and G.Jaberipur, “Energy efficient VLSI realization of binary-64 division with redundant number system,” IEEE Transaction on Very Large Scale Integration (VLSI) Systems, vol. PP, no. 99, pp. 1-8, 2016

[5] S. Oberman and M. Flynn, “Design issues in division and other floating-point operations,” IEEE Trans. Comput., vol. 46, no. 2, pp. 154–161, Feb. 1997

[6] S. F. Oberman and M. Flynn, “Division algorithms and implementations,” IEEE transaction on Computers, vol. 46, no. 8, pp. 833-854, 1997

[7] J. Ebergen and N.Jamadagni, “Radix-2 division algorithms with an over-redundant digit set,” IEEE Transaction on Computers, vol. 64, no. 9, pp. 2652-2663, 2015?? (2014)

[8] M. Flynn, “On division by functional iteration,” IEEE Transaction on Computers, vol. C-19, no. 8, pp. 702-7-6, 1970

[9] V. Gupta, D. Mohapatra, A.Raghunathan, and K. Roy, “Low power digital signal processing using approximate adders,” IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, vol. 32, no. 1, pp. 124-137, 2013

[10] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram, “RoBa Multiplier: A Rounding-Based Approximate Multiplier for High-Speed yet Energy-Efficient Digital Signal Processing,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 2, pp. 393-401, 2017

[11] S. Vahdat, M. Kamal, A. Afzali-Kusha, M. Pedram, and Z. Navabi, “TruncApp: A truncation-based approximate divider for energy efficient DPS applications,” Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, pp. 1635-1638

[12] L. Chen, J. Han, W. Liu, and F. Lombardi, “Design of approximate unsigned integer non-restoring divider for inexact computing,” Proceedings of the 25th edition on Great Lakes Symposium on VLSI, 2015, pp. 51-56

[13] D. Ferrari, "A division method using a parallel multiplier," IEEE Trans. Electronic Computers, vol. EC-16, pp. 224-226, 1967.

[14] L.Wu, and C. C. Jong, “A curve fitting approach for non-iterative divider design with accuracy and performance trade-off,” New Circuits and Systems Conference (NEWCAS), 2015, pp. 1-4

[15] Nangate 45nm Open Cell Library. http://www.nangate.com/.

[16] U.Qidwai and C. H. Chen, “Digital Image Processing: An algorithmic approach with MATLAB”, CRC Press, 2009

[17] The USC-SIPI Image Database [Online]. Available: http://sipi.usc.edu/database

[18] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: from vector visibility to structural similarity”, IEEE Transaction on Image Processing, vol. 13, no. 4, April 2014, pp. 600-612