Approximate Integer and Floating-Point Dividers with Near-Zero Error Bias

Hassaan Saadat
The University of New South Wales
Sydney, Australia
h.saadat@unsw.edu.au

Haris Javaid
Xilinx Inc.
Singapore
harisj@xilinx.com

Sri Parameswaran
The University of New South Wales
Sydney, Australia
sri.parameswaran@unsw.edu.au

ABSTRACT

We propose approximate dividers with near-zero error bias for both integer and floating-point numbers. The integer divider, INZeD, is designed using a novel, analytically deduced error-correction method in an approximate log based divider. The floating-point divider, FaNZeD, is based on a highly optimized mantissa divider that is inspired by INZeD. Both of the dividers are error configurable.

Our results show that the INZeD dividers have error bias in the range of $0.01 - 4.4\%$ with area-delay product improvement of $25\times - 95\times$ and power improvement of $4.7\times - 15\times$ when compared to the accurate integer divider. Likewise, compared to IEEE single-precision floating-point divider, FaNZeD dividers offer up to $985\times$ area-delay product and $77\times$ power improvements with error bias in the range of $0.04 - 2.2\%$. Most importantly, using our FaNZeD dividers, floating-point arithmetic can be more resource-efficient than fixed-point arithmetic because most of the FaNZeD dividers are even smaller and have better area-delay product than the 8-bit and 16-bit accurate integer dividers. Finally, our dividers show negligible effect on the output quality when evaluated with AlexNet and JPEG compression applications.

1 INTRODUCTION

Most of the modern compute-intensive applications such as audio and video encoding/decoding, neural networks, etc. can inherently tolerate controlled errors in the underlying computations without drastic degradation of the output quality [5, 6]. This error-resiliency can be exploited by using approximate arithmetic computations instead of the accurate ones to achieve better resource and energy efficiencies, typically known as approximate computing [15, 21].

A popular approach of approximate computing at the hardware-level is to trim down and simplify the logic design of an arithmetic unit by relaxing the accuracy constraints, resulting in lower resource consumption. Over the past decade, researchers primarily focused on approximation of adders and multipliers [9, 18]. However, in recent years, there is an increased interest in approximation of dividers [8, 11, 20]. In the approximation of arithmetic units, the following features are highly desirable:

- *Low error bias:* It leads to cancellation of errors instead of accumulation in successive computations [7, 18].
- *Error configurability:* It allows a designer to trade-off the error with resource-efficiency for a particular application because different applications are tolerant to different degrees of error [21].
- *Resource-efficient floating-point units:* Floating-point (FP) arithmetic, while being more resource-hungry, is advantageous over the alternative fixed-point representation of fractional numbers due to its ease-of-use. It saves the designer from the ordeals of overflow and underflow by offering a large dynamic range and bounded relative precision. Thus, approximate FP arithmetic that may be less precise but is more resource-efficient is typically preferable to fixed-point due to its benefit of ease-of-use [18].

DAC '19, June 2–6, 2019, Las Vegas, NV, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317773

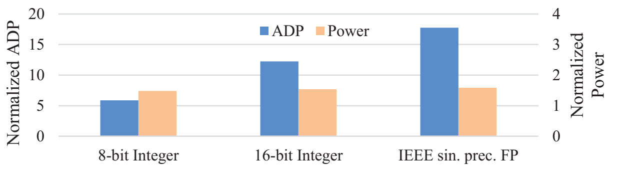

Figure 1: Resource consumption of division operation compared to multiplication operation.

While recent works on approximate dividers [8, 11, 20] provide error configurability, they lack the feature of low error bias. Moreover, these works did not target approximate FP dividers. In this paper, we address these limitations by proposing not only approximate integer dividers but also floating-point dividers, both with *near-zero* error bias and error-configurability.

Motivation: Typically, dividers are avoided because they are highly resource-hungry when compared to multipliers and adders. Nonetheless, they cannot always be avoided, and thus incur significant resource penalty when required. To demonstrate this, Fig. 1 presents the area-delay product (ADP) and power of the dividers for IEEE single-precision FP, 16-bit integer and 8-bit integer divisions, normalized with respect to their respective multiplication counterparts. The graph shows that the resource consumption of the dividers is significantly more than that of the multipliers. Therefore, resource-efficient dividers are highly desirable when they cannot be avoided due to application needs.

Contributions: In this paper, we aim for resource-efficiency of dividers through approximation with a particular focus on the above-mentioned features. Highlights of our contributions are:

- We present INZeD, a new approximate unsigned integer divider with near-zero error bias. It is designed by augmenting a novel, analytically deduced error-correction method with the classical linearly approximated log based integer divider.
- Next, we propose a set of FP dividers, FaNZeD, that are inspired by INZeD. An optimized version of INZeD is ingeniously used as the mantissa divider to create FP dividers that are even more resource-efficient than fixed-point dividers.
- Finally, we present error-configurable versions of both INZeD and FaNZeD, where a designer can choose the most optimized divider based on the application requirements.

Our experiments show that INZeD is capable of providing near-zero error bias, i.e., $\leq 0.02\%$ error bias with $34.6\times$ ADP and $5.6\times$ power improvement for 16-bit integer division. On the other hand, FaNZeD can offer up to $568\times$ ADP and $42\times$ power improvement when compared to IEEE single precision FP divider, with less than 0.1% error bias. Our results further show that INZeD dividers contribute to the Pareto front of the design space of state-of-the-art approximate dividers, while FaNZeD dividers offer better trade-offs than traditional precision scaling. Results show that our proposed FaNZeD dividers can be even more resource-efficient than their fixed-point counterparts. Finally, application-level evaluation of the proposed dividers demonstrates minuscule degradation of the output quality.

Organization: The related work and preliminaries are covered in Section 2 and 3. The analytical analysis and derivation of near-zero error bias method, and the approximate integer divider INZeD are presented in Section 4. The approximate FP divider FaNZeD is explained in Section 5. Section 6 presents the experimental setup and results. Finally, the paper is concluded in Section 7.

2 RELATED WORK

Most of the existing work on approximate arithmetic units focuses on adders and multipliers. Details can be found in [9, 10, 18]. Nonetheless, in recent years, approximation of dividers has also become an active research topic.

A class of approximate integer dividers is based on using approximate adder or subtracter cells in $2N$-by-N array dividers [16]. In [1] and [2], the accurate divider cells in a $2N$-by-N non-restoring and restoring array divider, respectively, are truncated or replaced by approximate divider cells. These approximate divider cells are built using approximate full-subtracters. In [3, 4], an approximate signed-digit adder cell is designed and then used in combinational implementation of higher radix SRT divider. It is reported in [11] that the resource improvements offered by these dividers are small.

A wider class of approximate division schemes propose approximation at the algorithmic/architectural level. Some dividers [19, 22] employ look-up tables in their approximation schemes, however, the area and power consumption of these dividers are substantial due to the presence of look-up tables [11]. In TruncApp [20] approximate inverse of the divisor is determined and multiplied with the truncated dividend. TruncApp-AM (TAM) is a variation of this design in which the multiplication is also approximated.

The DAXD design [8] consists of a dynamic selection of bits in a 2N-by-N divider depending on the position of the leading-one and then uses a smaller accurate sub-divider. This divider suffers from large error due to overflow problem, as identified by [11]. The authors in [11] also propose a solution to the overflow problem: they use a larger sub-divider in their $2N$-by-N AAXD divider design.

The classical approximate log based divider (ALD) [14] computes log of the operands through linear approximation and then uses the log-division property to perform division. This division scheme is efficient but it is non-configurable and suffers from high error bias as all error values have the same sign. A hybrid design (AXHD) [13] combines an array divider and the ALD divider. This divider does not propose any modification in the ALD design and thus it also suffers from high error bias.

Our proposed dividers are inspired by the work in [18] and differ from all the above works in two major aspects: (1) a novel low-overhead error-correction method to achieve near-zero error bias is introduced; (2) both integer and FP dividers are targetted.

3 PRELIMINARIES

The proposed approximate dividers are based on the classical approximate log based divider (ALD) proposed by Mitchell [14], so we first present its mathematical formulation. An N-bit unsigned integer X can be represented as:

$$X = \sum_{i=0}^{N-1} 2^i b_i = 2^{k_x} + \sum_{i=0}^{k_x-1} 2^i b_i = 2^{k_x}\left(1 + \sum_{i=0}^{k_x-1} 2^{i-k_x} b_i\right)$$

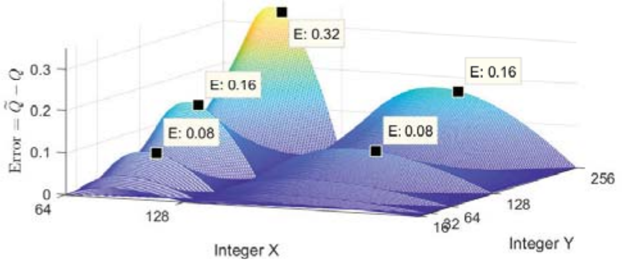

Figure 2: Error profile of the approximate log based divider (ALD).

where b_i is the i-th bit and k_x is the position of the leading-one, such that $0 \le k_x \le (N-1)$. Let $x = \sum_{i=0}^{k_x-1} 2^{i-k_x} b_i$, then the binary log of X can be linearly approximated as:

$$log_2'(X) = k_x + x \tag{1}$$

where k_x is the characteristic part of the approximate log and $0 \le x < 1$ is the fractional part.

For division, suppose there are two N-bit integers X and Y, with their leading ones at k_x and k_y respectively. Then, the approximate binary log of quotient Q is:

$$log_2'(Q) = k_x - k_y + x - y \tag{2}$$

To obtain the approximate quotient, the inverse of the approximate binary log is applied:

$$Q' = \begin{cases} 2^{k_x-k_y-1}(2 + x - y) & x - y < 0 \\ 2^{k_x-k_y}(1 + x - y) & x - y \ge 0 \end{cases} \tag{3}$$

This expression is decomposed into two cases because the inverse of log requires the fractional part to be in the range [0,1). The first case happens when a borrow is taken from the characteristic part.

4 INZED: APPROXIMATE INTEGER DIVIDER

In this section, we first present our proposed near-zero error bias method. Then we present the hardware architecture of INZeD followed by an explanation of its error configurability mechanism.

4.1 Near-Zero Error Bias Method

The accurate quotient from the division of X and Y is $Q = 2^{k_x-k_y} \cdot ((1+x)/(1+y))$. Thus, the error in quotient can be computed as:

$$E_Q = Q' - Q = \begin{cases} 2^{k_x-k_y} \frac{(y + x(y-1) - y^2)}{2(1+y)} & x - y < 0 \\ 2^{k_x-k_y} \frac{(xy - y^2)}{(1+y)} & x - y \ge 0 \end{cases} \tag{4}$$

Fig. 2 plots this error. For better visualization, the plot is shown for $X = \{16, 17 \ldots 255\}$ and $Y = \{64, 65 \ldots 255\}$. From Eq. 4 and the graphical representation in Fig. 2, we make the following key observations: (1) The error always has the same sign, thus the classical log-based divider is highly biased. (2) Although the error has an increasing trend, the error profiles in each power-of-two-interval (each pair of k_x, k_y) are simply replicas of each other with a scaling a factor of $2^{k_x-k_y}$.

Based on these observations and our goal of reducing error bias with minimal overhead, we propose to subtract the average error within each power-of-two-interval from the approximate quotient values in that interval to achieve a near-zero average error. This average error, for a given k_x and k_y over $x \in [0,1)$ and $y \in [0,1)$ can be computed as follows:

$$Avg(E_Q) = \frac{1}{(1-0)(1-0)} \int_0^1 \int_0^1 E_Q dxdy$$

$$= 2^{k_x - k_y} \left(\int_0^1 \int_0^y \frac{(y + x(y-1) - y^2)}{2(1+y)} dxdy + \int_0^1 \int_y^1 \frac{(xy - y^2)}{(1+y)} dxdy \right)$$

$$= (0.043) \times 2^{k_x - k_y}$$

As expected, the mean error in each power-of-two-interval is a constant $\epsilon = 0.043$ scaled by $2^{k_x - k_y}$, thus our proposed division method can mathematically be expressed as:

$$Q' = \begin{cases} 2^{k_x - k_y - 1}(2 + x - y - 2\epsilon) & x - y < 0 \\ 2^{k_x - k_y}(1 + x - y - \epsilon) & x + y \geq 0 \end{cases} \tag{5}$$

We subtract ϵ before scaling of the fractional part to avoid the extra scaling hardware (see Section 4.2). Our proposed near-zero error bias method consists of a single error-correction term which is independent of the inputs as well as the size of the divider N. As we will demonstrate in Section 6.3, our near-zero error bias method incurs a small resource overhead when compared to the classical log-based divider but results in a much improved error behavior.

4.2 Hardware Architecture

As mentioned in Section 2, most of the previous works on approximate integer dividers are based on $2N$-by-N dividers. Therefore, we also present the architecture of a $2N$-by-N divider for our near-zero error bias method; however, our proposal is equally applicable to N-by-N dividers.

Approximation of the error-correction term: The error-correction term determined in the previous section needs a large number of bits for accurate representation, which would require the use of a large subtracter. Instead, we approximate this term in 8-bits as $\epsilon = 2^{-5} + 2^{-8}$, so that only an 8-bit subtracter and an 8-bit 2x1 multiplexer (for left shift) are required, thus keeping the overhead to a minimum.

Explanation of the design: The logic design of the proposed divider is shown in Fig. 3. First, the approximate logs (Equation 1) of the $2N$-bit dividend (X) and N-bit divisor (Y) are computed using leading-one detectors (LOD) and barrel shifters of appropriate sizes. Specifically, the LOD produces the characteristic part of the log value, while the barrel shifter selects the bits next to the leading-one to produce the fractional part (and inserts extra zeros to the right to keep bit-width constant). The fractional part is appended to the right of the characteristic part, effectively implementing Equation 1.

The two approximate log values are then subtracted using the main subtracter. Next step is the subtraction of the error-correction term ϵ from the fractional part of the log difference. The error-correction term is left shifted if a borrow was taken from the characteristic part in the main subtracter. Next, a '1' or '0' is appended to the left of the result if there is a borrow out in this second subtraction. This result is then scaled using another barrel shifter with respect to the characteristic part from the main subtracter (which is $2^{k_x - k_y}$ or $2^{k_x - k_y - 1}$).

Optimization of bit-widths: In a standard $2N$-by-N divider, the quotient is N-bit. To avoid overflow, the condition $X < 2^N Y$ must be satisfied [16], which means that the final scaling in our divider ranges from 2^0 to 2^{N-1}. Since we are dealing with an integer divider, any part of the expression that remains in the fractional part after scaling needs to be ignored. Thus, for any possible output, we need a maximum of $N - 1$ bits in the fractional part of the expression before the final scaling, which means that only $N - 1$ bits are needed from both subtracters. We truncate N lsb bits from $log'(X)$, leaving $N - 1$ bits in the fractional part. This reduces the logic size of the barrel shifter for dividend X, the main subtracter ($log(2N) + N - 1$-bits), and the final barrel shifter. Note that these optimizations do not affect the accuracy of the divider because these truncated bits do not contribute to the final quotient. Fig. 3 reports the number of bits at different stages of our divider.

An exception here is when $N \leq 8$ (e.g. 16-by-8 divider) because the error-correction term has 8 bits in the fractional part. Specifically for the 16-by-8 divider, we keep N bits in the fractional part of $log'(X)$ and append an extra zero to the right of $log'(Y)$. In this case, the fractional part of the main and second subtracter would be N-bits wide, and the input to the final barrel shifter would be $N + 1$ bits when the inverted r_o is appended. However, the extra least significant bit will be dropped by the barrel shifter after scaling, keeping the quotient to N-bits.

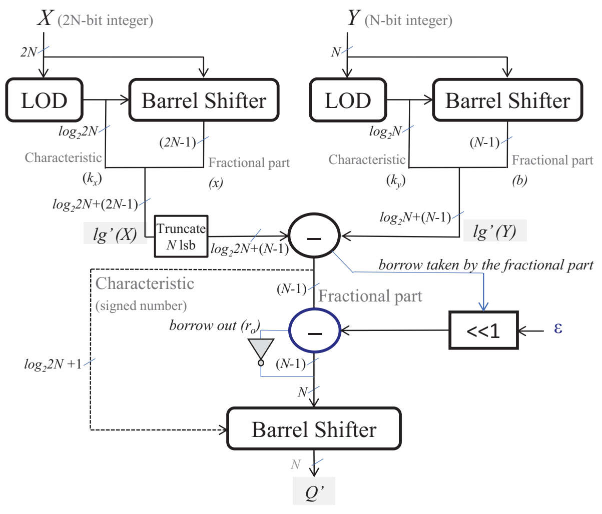

Figure 3: The proposed approximate integer divider INZeD.

4.3 Error Configurability

We introduce error configurability in the proposed divider by truncating t least significant bits from the inputs of the main subtracter. This enables reduction in the size of the input barrel shifters, the main subtracter, as well as the final barrel shifter, while introducing more error due to the loss of bits. When $t \geq N - 8$, the number of bits in the fractional part are < 8. In such a scenario, we also truncate $t - (N-1) + 8$ bits from the second input of the error-correction subtracter. When the number of bits in the fractional part from the output of the last subtracter is lower than $N - 1$, we append zeros to its right before the final scaling. The error configurable design is referred to as INZeD-t in this paper.

For $N = 8$, i.e., in case of 16-by-8 divider, we used an extra bit as explained in the previous subsection, keeping the fractional part to N bits. Therefore, truncating 1 bit ($t = 1$) means truncating that extra bit, and so on. Specifically, we will have $N - 1 + 1 - t = N - t$ bit main subtracter.

The truncation of bits may introduce the problem of overflow in the $2N$-by-N divider. Specifically, the characteristic from the main subtracter may have the value N instead of $N - 1$, which consequently results in a final scaling by a factor of 2^N. This causes the output quotient to become larger than what can be represented in N-bits, resulting in overflow. To handle this, we put a comparator

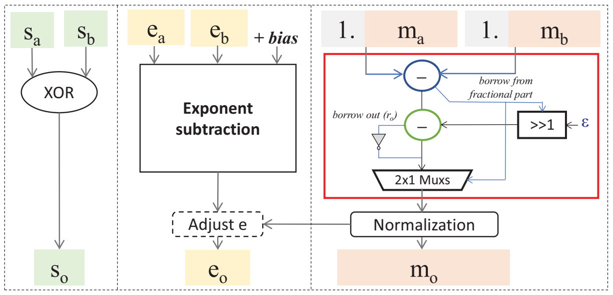

Figure 4: The proposed approximate FP divider FaNZeD. The optimized mantissa divider reduces to extremely simple logic.

in the design that checks if the characteristic is N. In such a case, the output quotient is set to the maximum possible value of 2^N-1.

5 FANZED: APPROXIMATE FP DIVIDER

In an IEEE single-precision FP format, a number consists of a sign bit (s), an exponent (e), and a mantissa (m). The mantissa is normalized, so its first bit is always '1', which is the hidden bit and not stored with the number. The FP division consists of mantissa division, exponent subtraction, rounding, normalization, and some logic for exception handling.

To design our proposed approximate FP divider: FaNZeD, we make two key observations. (1) The synthesis of an IEEE single-precision FP divider with restoring array divider for mantissa division using Cadence RTL Compiler reported that the mantissa divider consumes more than 92% area and 95% power of the entire FP divider. (2) The mantissa division is essentially an unsigned division (with additional zeros assumed to the right of the numerator to obtain the fractional bits). Based on these observations, we applied the following optimizations to INZeD, which naturally led us to use it as a resource-efficient mantissa divider. (1) Since the positions of the leading-ones in the mantissas are already known, the LODs and barrel shifters at the inputs can be removed. This optimization further allows to replace the output barrel shifter with a simple multiplexer and reduce the size of the main subtracter. (2) The overflow check mentioned in the last section is not required, because the characteristic part never becomes equal to N. Moreover, in the proposed approximate FP divider, the rounding unit is removed because fine-precision is ignored. The exponent subtraction, normalization and exception handling modules are kept the same. The error-correction term ϵ is approximated as $0.043 \approx 2^{-5} + 2^{-7}$. This approximation is empirically fine-tuned to achieve best error bias with low overhead. Fig. 4 shows the overall hardware architecture of FaNZeD. For error configurability, we vary the parameter t in the mantissa divider to create differing FP dividers, allowing trade-off of error with resource-efficiency. The error configurable version is referred to as FaNZeD-t in this paper. Although we derive FaNZeD-t from single-precision, our proposal is equally applicable to half or double-precision floating-point formats.

6 EXPERIMENTS AND RESULTS

6.1 Comparison with State-of-the-art

We compare our integer dividers (INZeD-t) with the accurate and state-of-the-art approximate integer dividers: DAXD [8], AAXD [11], TruncApp [20], TruncApp-AM (TAM) [20], AXHD [13], and the approximate log-based divider (ALD) [14]. Similar to most previous works, we use an array divider design for the accurate divider.

The proposed approximate FP dividers (FaNZeD-t) are compared with traditional precision scaling method in IEEE single-precision FP dividers, where t bits from both the mantissas are truncated. We apply precision scaling both with and without the rounding unit, essentially creating two versions of the IEEE FP divider, referred to as IEEE-R and IEEE-NR in this paper, respectively.

Table 1: Results for the proposed INZeD dividers.

Divider	Error Bias (%)	Mean Error (%)	Area (μm^2)	Power (μW)	Freq. (GHz)	ADP ($\mu m^2 \times ns$)
32-by-16 Divider ($N = 16$)						
Accurate	-	-	2980.9	397.9	0.16	18869
ALD [14]	3.91	3.91	1334.8	75.4	2.11	632.7
INZeD	−0.02	2.74	1389.3	84.9	1.91	728.0
INZeD-2	−0.02	2.74	1315.9	79.7	1.93	680.3
INZeD-4	−0.01	2.74	1327.9	80.5	2.09	636.1
INZeD-6	−0.01	2.74	1164.9	70.6	2.14	544.0
INZeD-8	0.10	2.76	966.2	57.5	2.18	442.5
INZeD-10	0.47	2.74	778.0	43.6	2.38	327.6
INZeD-11	1.43	3.42	763.9	40.0	2.43	314.7
INZeD-12	4.04	4.96	634.3	30.8	2.53	251.2
INZeD-13	4.42	7.16	541.2	26.8	2.72	198.6
16-by-8 Divider ($N = 8$)						
Accurate	-	-	810.3	67.5	0.58	1408
ALD [14]	3.93	3.93	600.7	34.4	2.61	230.1
INZeD	0.001	2.75	724.9	44.4	2.46	294.3
INZeD-1	0.11	2.77	731.7	45.2	2.53	289.0
INZeD-2	0.52	2.69	626.8	36.3	2.55	245.7
INZeD-3	0.64	2.74	592.5	32.8	2.62	225.7
INZeD-4	1.76	3.52	542.1	28.3	3.07	176.7
INZeD-5	4.50	5.29	416.8	21.1	3.36	124.2

6.2 Implementation Details

For fairness of comparison, all the dividers are implemented as single cycle combinational circuits. Each divider is coded in Verilog HDL and synthesized with Cadence RTL Compiler using TSMC 45nm standard-cell library at the maximum possible frequency. We placed sequential elements at the inputs and outputs of a divider for timing constraints, however, we only used the combinational logic area and power when reporting the results.

For INZeD, we implemented 16-by-8 and 32-by-16 dividers. The accurate integer divider, the sub-dividers in DAXD and AAXD, and the mantissa divider in IEEE FP dividers are implemented as restoring array dividers [16]. The multiplier in TruncApp is implemented as the Wallace tree multiplier. The bit-width optimization described in Section 4.2 is also applied to ALD for fair evaluation of the overhead of the proposed near-zero error bias method.

For error analysis, behavioral models of the dividers are implemented in MATLAB. The functional equivalence of each model is verified by comparing the output of the Verilog HDL simulations with that of the MATLAB simulations for the same set of random inputs. For 32-by-16 dividers, we performed Monte Carlo simulations with 67 million inputs uniformly distributed in the range of $X = \{1....2^{2N} - 1\}$ dividend and $Y = \{1...2^N - 1\}$ divisor. Only those values are selected where no overflow occurs in the accurate divider, i.e., $X < 2^N Y$ [16]. For 16-by-8 dividers, exhaustive simulations were performed over the same choice of inputs. For FP dividers, 5 million random inputs from the interval [1,2) are used. Like previous works [18, 23], this interval is chosen because the exponent part is not changed and hence the error analysis does not get affected.

Two popular error metrics, mean of absolute relative error (mean error) and mean of relative error (error bias) are used to measure the efficacy of the proposed dividers. The reference for integer dividers is accurate division, and that for FP dividers is the IEEE single-precision division output. In some approximate integer dividers, the approximate output can be non-zero even when accurate output is zero, which results in an infinite relative error. We assign a penalty of 100% error for pragmatic error analysis. If both the accurate and approximate values are zero, then we set the relative error as 0%.

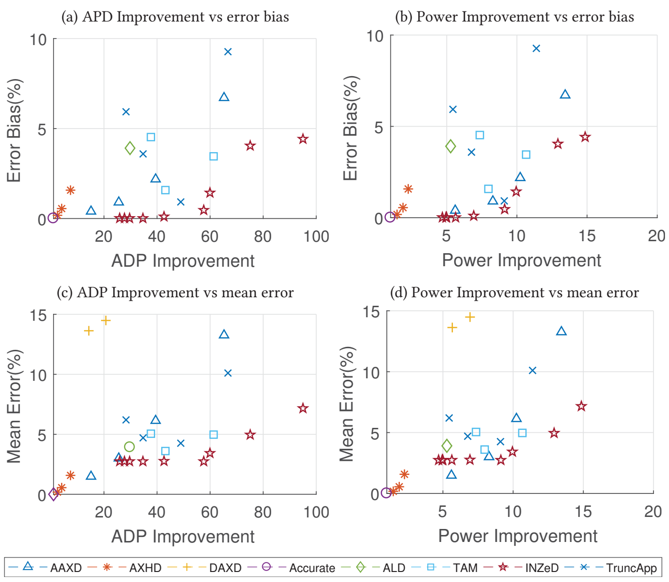

Figure 5: 16-bit divider design space. INZeD dividers are part of the Pareto front.

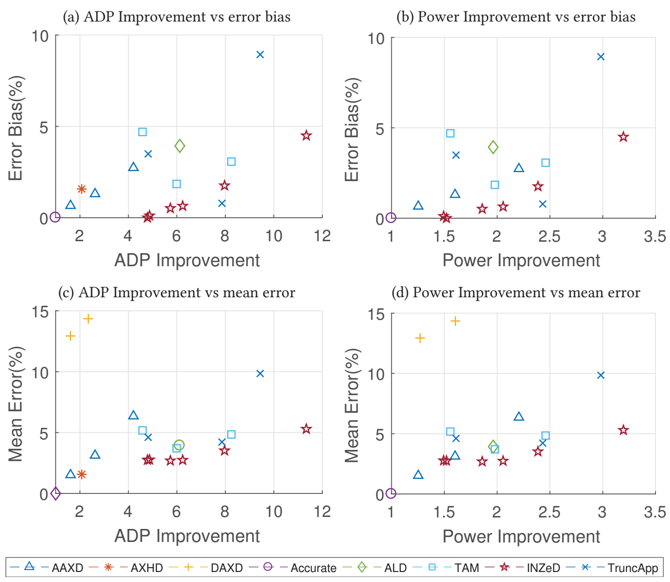

Figure 6: 8-bit divider design space. INZeD dividers are part of the Pareto front.

6.3 INZeD Results

ALD comparison: Table 1 reports the results for INZeD, ALD, and the accurate dividers. The error bias for INZeD divider is 0.02% for N=16 and 0.001% for N=8. This error bias is orders of magnitude smaller than the error bias of ALD (3.9%), while at the same time, mean error is improved from 3.9% to 2.7%. In comparison to the accurate divider, INZeD offers significant ADP improvements: 25.9× and 4.8× for $N = 16$ and $N = 8$ respectively (rows 1 and 3).

Error configurability: Table 1 also lists the results for error configurability in INZeD-t dividers. As expected, the resource footprint reduces at the cost of more error. For N=16, the parameter t manifests increased errors when $t \geq 8$ because at this point, the truncation of the 8-bit error-correction term starts. The INZeD-t dividers are able to provide 57.6× ADP and 9.1× power improvement for less than 0.5% error bias (rows 1 and 8). For N=8, the increased error trend is observed at the very start, i.e. $t \geq 1$, again because of the truncation of the error-correction term. Nonetheless, INZeD-t is able to provide 5.7× ADP improvement for nearly 0.5% error bias (rows 1 and 5).

It is noteworthy that the INZeD-t dividers can be even more efficient than the ALD divider while still providing lower error. Specifically, this happens beyond $t \geq 6$ for N=16 and $t \geq 3$ for N=8.

State-of-the-art comparison: Fig. 5 and Fig. 6 compare INZeD-t dividers with the state-of-the-art approximate integer dividers for $N = 16$ and $N = 8$, respectively. The error metrics are plotted on the y-axis while the improvement metrics are plotted on the x-axis.

In Fig. 5 (a, b) for $N = 16$, it can be observed that the INZeD-t dividers offer lower error bias than all the other state-of-the-art approximate dividers, and are part of the Pareto front. With an error bias range of 0.01 − 4.4%, our dividers provide ADP improvements in the range of 25 − 95× and power improvements in the range of 4.7 − 15×. Fig. 5 (c, d) demonstrates the efficacy of INZED-t dividers in terms of mean error. Again, the Pareto front includes many points from our INZeD-t dividers. For $N = 8$, Fig. 6 depicts similar observations.

Table 2: Results for the proposed FaNZeD dividers.

Divider	Error Bias (%)	Mean Error (%)	Area (μm^2)	Power (μW)	Freq. (GHz)	ADP ($\mu m^2 \times ns$)
IEEE FP	0	0	7584.7	1343.0	0.06	123289
FaNZeD-0	-0.04	2.89	904.6	58.2	2.21	409.8
FaNZeD-10	-0.03	2.89	805.8	47.5	2.24	360.2
FaNZeD-15	-0.04	2.89	558.1	31.9	2.48	225.5
FaNZeD-18	-0.36	2.97	472.8	28.6	2.54	186.3
FaNZeD-19	-0.86	2.89	386.4	22.2	2.67	144.5
FaNZeD-20	-2.22	3.81	363.9	17.4	2.91	125.2

6.4 FaNZeD Results

Error configurability: The results for FaNZeD-t dividers are reported in Table 2 and Fig. 7. From the table, it can be seen that our dividers offer up to 985× ADP and 77× power improvements compared to the IEEE single-precision divider (rows 1 and 7). In Fig. 7 (a,b), the FaNZeD-t dividers exhibit near-zero error bias (< 0.1%) for nearly 600× ADP improvement and 45× power improvement. The error

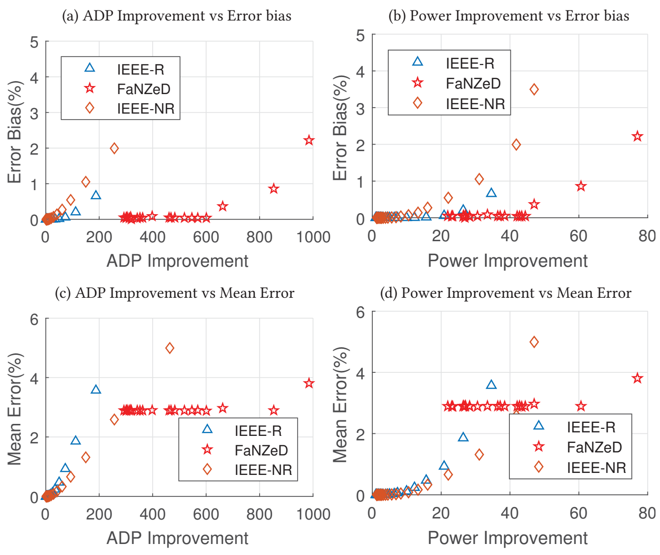

Figure 7: **Floating-point divider design space. FaNZeD dividers are part of the Pareto front. IEEE-R and IEEE-NR mean precision scaling with and without rounding, respectively.**

Table 3: AlexNet classification results using different dividers.

Divider		IEEE FP	FaNZeD-t t=0	t=10	t=15	t=18	t=20
Improvement	ADP	-	300 ×	342 ×	546 ×	661 ×	985 ×
	Power	-	23.1 ×	28.3 ×	42.1 ×	47.0 ×	77.0 ×
Error Rate (%)	Top-1	43.42	43.50	43.50	43.50	43.40	43.24
	Top-5	20.64	20.68	20.68	20.68	20.60	20.66

bias increases to 2.2% for higher values of improvements. In contrast, the error bias of traditional precision scaling with and without rounding shoots up for even less than 100× ADP improvement. Although the mean error of FaNZeD-*t* dividers is higher (> 2.9%) than most of the precision scaling dividers, they offer much higher ADP and power improvements than the later, and thus constitute the extended Pareto front as shown in Fig. 7 (c,d).

Fixed-point comparison: Since fixed-point arithmetic is implemented with integer dividers, our results show that FaNZeD-*t* dividers enable FP arithmetic which can be even more resource-efficient than fixed-point arithmetic. For example, FaNZeD-10 (from Table 2) is smaller than both 16-bit accurate and INZeD dividers, as well as smaller than 8-bit accurate integer divider (from Table 1).

6.5 AlexNet Application

AlexNet is a convolutional neural network (CNN) used for image classification [12]. While it is mostly dominated by multiply-and-add operations, it also contains division operations. Specifically, there are two *Local Response Normalization* layers and one *softmax* layer which require division operation. The local response normalization layer is important as it reduces Top-1 and Top-5 error rates by 1.4% and 1.2%, respectively [12], and involve nearly 0.5 million division operations in the AlexNet.

We implemented AlexNet, pretrained on ILSVRC dataset [17], with IEEE single-precision FP arithmetic in software. We then replaced each division operation in all the layers with the software model of FaNZeD-*t* dividers. We performed classification of five thousand images from the ILSVRC 2012 validation set, and compared the classification accuracy. The results reported in Table 3 show that FaNZeD-*t* dividers have a negligible impact on the accuracy of AlexNet, while offering more than 900× ADP improvement.

6.6 JPEG Compression Application

JPEG compression is a lossy image compression technique in which the compressed image suffers from quality loss. It provides the option of different quality specifications, where the quality of the compressed image is determined by comparing it with the original uncompressed image using a quantitative metric such as PSNR.

For achieving different quality compressions, the discrete cosine transform (DCT) coefficients are divided by a quality map (determined by the user-specified quality level). The DCT coefficients are real numbers, so we implemented JPEG compression using FP arithmetic and 16-bit fixed-point (integer) arithmetic. For both implementations, we used accurate and approximate versions (INZeD-*t* and FaNZeD-*t*) of dividers. Table 4 reports the results of JPEG compression on two 512x512-pixel grayscale images at three different quality levels (Q). Due to the limitation of space, results for only selected dividers are presented. It is evident that FaNZeD-*t* dividers provide similar image quality for all cases (maximum loss of 0.4dB) as the IEEE single-precision FP divider even at the maximum level of approximation ($t = 20$). More importantly, our FaNZeD-*t* dividers provide significantly higher ADP and power improvements (985× and 77× respectively) than the fixed-point dividers (both accurate and approximate versions), while providing similar (or even better) image quality.

Table 4: JPEG quality results (in PSNR) using different dividers.

Divider	**Improvement** ADP	Power	**Image: Mandrill** Q: 25	Q: 50	Q: 75	**Image: Lena** Q: 25	Q: 50	Q: 75
Floating-Point								
IEEE sp FP	-	-	28.3	31.4	34.6	31.6	34.0	36.1
FaNZeD-20	985 ×	77.0×	28.3	31.5	34.5	31.6	33.9	35.7
Fixed-Point								
Acc. Integer	6.5×	3.4 ×	28.2	31.4	34.5	31.5	33.9	36.0
INZeD-0	15.8×	169×	28.0	31.1	34.1	30.9	33.3	35.2
INZeD-13	620×	50.2×	28.3	30.9	33.1	30.0	31.2	32.1

7 CONCLUSIONS

We proposed approximate dividers for integer and floating-point division. The integer divider INZeD is designed by combining a novel error-correction method with the classical approximate log based divider with the aim of near-zero error bias. Our results demonstrate that INZeD divider offers Pareto optimal points in the design space consisting of state-of-the-art approximate integer dividers. The floating-point divider FaNZeD is designed by using an optimized mantissa divider based on the INZeD divider. Our results show that a large set of FaNZeD dividers can be derived that offer better trade-offs than traditional precision scaling. Moreover, some of the FaNZeD dividers are even more resource-efficient than 16-bit and 8-bit fixed-point (integer) dividers. Application-level evaluations show that our dividers have minuscule degradation of the output quality.

REFERENCES

[1] L. Chen et al. 2015. Design of Approximate Unsigned Integer Non-restoring Divider for Inexact Computing. In *Proc. 25th GLSVLSI '15*. 51–56.
[2] L. Chen et al. 2016. On the Design of Approximate Restoring Dividers for Error-Tolerant Applications. *IEEE Trans. Comput.* 65, 8 (2016), 2522–2533.
[3] L. Chen et al. 2017. Design of Approximate High-Radix Dividers by Inexact Binary Signed-Digit Addition. In *Proc. 27th GLSVLSI '17*. 293–298.
[4] L. Chen et al. 2018. Design, Evaluation and Application of Approximate High-Radix Dividers. *IEEE Trans. Multi-Scale Computing Systems* 4, 3 (2018), 299–312.
[5] V. K. Chippa et al. 2013. Analysis and Characterization of Inherent Application Resilience for Approximate Computing. In *Proc. 50th DAC '13*. 113:1–113:9.
[6] H. Esmaeilzadeh et al. 2012. Architecture Support for Disciplined Approximate Programming. In *Proc. 17th ASPLOS '12*. 301–312.
[7] S. Hashemi et al. 2015. DRUM: A Dynamic Range Unbiased Multiplier for Approximate Applications. In *Proc ICCAD '15*. 418–425.
[8] S. Hashemi et al. 2016. A low-power dynamic divider for approximate applications. In *In Proc. 53rd DAC*. 1–6.
[9] H. Jiang et al. 2015. A Comparative Review and Evaluation of Approximate Adders. In *Proc. 25th GLSVLSI '15*.
[10] H. Jiang et al. 2016. A comparative evaluation of approximate multipliers. In *Proc. NANOARCH '16*. 191–196.
[11] H. Jiang et al. 2018. Adaptive Approximation in Arithmetic Circuits: A Low-power Unsigned Divider Design. In *Proc. DATE '18*. 1411–1416.
[12] Alex Krizhevsky et al. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.
[13] W. Liu et al. 2018. Combining Restoring Array and Logarithmic Dividers into an Approximate Hybrid Design. In *Proc. 25th ARITH '18*. 92–98.
[14] J. N. Mitchell. 1962. Computer Multiplication and Division Using Binary Logarithms. *IRE Trans. on Electronic Computers* EC-11, 4 (1962), 512–517.
[15] Sparsh Mittal. 2016. A Survey of Techniques for Approximate Computing. *ACM Comput. Surv.* 48, 4, Article 62 (March 2016), 33 pages.
[16] Behrooz Parhami. 2000. *Computer Arithmetic: Algorithms and Hardware Designs.* Oxford University Press, Inc., New York, NY, USA.
[17] O. Russakovsky et al. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252.
[18] H. Saadat et al. 2018. Minimally Biased Multipliers for Approximate Integer and Floating-Point Multiplication. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* 37, 11 (2018), 2623–2635.
[19] M. Vaeztourshizi et al. 2018. An Energy-Efficient, Yet Highly-Accurate, Approximate Non-Iterative Divider. In *Proc. ISLPED '18*. 14:1–14:6.
[20] S. Vahdat et al. 2017. TruncApp: A truncation-based approximate divider for energy efficient DSP applications. In *Proc. DATE '17*. 1635–1638.
[21] S. Venkataramani et al. 2015. Approximate Computing and the Quest for Computing Efficiency. In *Proc. 52nd DAC '15*. 120:1–120:6.
[22] R. Zendegani et al. 2016. SEERAD: A high speed yet energy-efficient rounding-based approximate divider. In *Proc. DATE '16*. 1481–1484.
[23] H. Zhang et al. 2014. A low-power accuracy-configurable floating point multiplier. In *Proc. 32nd ICCD '14*. 48–54.