FPAD: A Multistage Approximation Methodology for Designing Floating Point Approximate Dividers

Chandan Kumar Jha[*1], Kailash Prasad[*1], Vibhor Kumar Srivastava[+2], and Joycee Mekie[1]
Electrical Engineering Department, Indian Institute of Technology Gandhinagar, India[1]
Computer Science and Engineering Department, IITDM Kurnool, India[2]
Email: {chandan.jha, kailash.prasad, coe16b032@iiitk.ac.in, and joycee}@iitgn.ac.in

***Abstract*—Approximate computing has emerged as a unique proposition for error-resilient applications such as image/video processing, neural networks, and the like, where both performance and power can be simultaneously reduced by trading off output quality. In this paper we propose a multistage approximation methodology for designing IEEE 754 floating point approximate dividers (FPADs). We propose a number of FPADs with varying upper bounds on error. For the same mean error, FPAD is $2.84\times$ better in terms of power-delay product (PDP) as compared to state of the art approximate floating point divider. Further, when applied on applications, such as image enhancement, mean filtering and JPEG compression, FPAD outperforms the existing state-of-the-art approximate divider in terms of PDP by 25%. We also show that FPAD gives same PDP benefits in Alexnet convolutional neural network without noticeable drop in top-5 and top-1 accuracy.**

I. Introduction

Approximate computing exploits error resilience of applications to gain benefits in energy, performance, and area [1]–[4]. The approximation can be introduced at various levels of the computing stack ranging from applications to circuits [5]–[10]. In this work, we focus on introducing approximation at the circuit level [11]–[13]. We focus on the design of approximate floating point divider as it is one of the most power hungry arithmetic unit [14]–[17]. Approximate arithmetic circuits, based on integer and floating point, adders, subtractors, and multipliers have been extensively explored [18]–[23]. Approximate integer dividers have also received significant attention [24]–[28], but very few works have focused on the design of approximate floating point dividers [15], [17].

We propose a methodology for the design of floating point approximate divider (*FPAD*) based on a mathematical approach. *FPAD* requires only shift and add operations. A floating point number consists of three parts, a *sign* bit, *exponent*, and *mantissa*. The *mantissa* division is the most power consuming part of a floating point divider [17]. Thus the existing works have focused on reducing the complexity of the *mantissa* division [15], [17]. In *CADE* [15], *mantissa* division is performed by subtracting the *mantissa*'s, followed by another subtraction from a wide set of pre computed offset values. The pre computed values need to be stored in lookup tables (LUTs), which consumes significant power and area. In *FaNZeD* [17], the *mantissa* bits are subtracted similar to [15], but rather than having overheads in terms of implementing LUTs, a fixed error correction term is used to reduce the error. However, the maximum error can reach up to 11% in *FaNZeD*. Unlike previous works, *FPAD* design methodology can be used to design a wide range of approximate floating-point dividers. This can help the users to design tailored approximate dividers depending upon the application's requirements. *FPAD* also consumes lesser power as compared to the state of the art designs while maintaining a maximum bounded error value. The following are the contributions of our work.

- We propose a multistage approximation methodology for the design of floating point approximate dividers *(FPAD)*.
- Every *FPAD* design has a bounded maximum error. We also evaluated truncated FPAD designs.
- *FPAD* outperforms the existing state of the art design by $2.84\times$ in terms of power delay product (PDP).
- *FPAD* consumes 25% lesser PDP as compared to state of the art designs in image processing and neural network applications while maintaining same output quality.

II. Multistage Approximation Methodology

In single precision IEEE standard for floating-point arithmetic *(SP IEEE 754)* [29], the number consists of 3 parts: *sign*, *exponent*, and *mantissa* bits. The first bit represents the *sign*, and the next 8 bits make the *exponent*, and the last 23 bits are the *mantissa* bits. The value of *mantissa* always ranges between 1 and 2. In a *SP IEEE 754* divider, *mantissa* division constitutes of 92% area and 95% power [17]. Thus, approximation is introduced in the *mantissa* division as it is the most power consuming part of the division. Also, introducing approximation in *exponent*s or *sign* bits leads to large errors.

In this work, *mantissa* division is approximated using only shift and add operation. If A and B are two numbers in *SP IEEE 754* representation and X and Y are their respective *mantissa's* then *mantissa* division can be mathematically formulated in the following way :

$$X = 1.x_{23}x_{22}x_{21}\dots x_1, Y = 1.y_{23}y_{22}y_{21}\dots y_1 \quad (1)$$

$$Z = \frac{X}{Y} = X * (\frac{1}{Y}) = (X \gg p_1) + (X \gg p_2) + \dots \quad (2)$$

where

$$\frac{1}{Y} = (1 \gg p_1) + (1 \gg p_2) + \dots \quad (3)$$

[*] Equal Contribution
[+] This work was done when Vibhor K Srivastava was at IIT Gandhinagar

978-1-7281-3320-1/20/$31.00 ©2020 IEEE

TABLE I: FPAD design approximation for various levels and within each level using different number of adder units for approximation

Level	Y	approx$(1/Y)$	0 Adder	1 Adder	2 Adders	3 Adders
1	0	≤ 1	$Z = X$	$Z = X$	$Z = X$	$Z = X$
	1	≤ 0.66	Z = X≫1	Z = X≫1 + X≫3	Z = X≫1 + X≫3 + X≫5	Z = X≫1 + X≫3 + X≫5 + X≫6
2	00	≤ 1	$Z = X$	$Z = X$	$Z = X$	$Z = X$
	01	≤ 0.80	Z = X	Z = X≫1 + X≫2	Z = X≫1 + X≫2 + X≫5	Z = X≫1 + X≫2 + X≫5 + X≫6
	10	≤ 0.66	Z = X≫1	Z = X≫1 + X≫3	Z = X≫1 + X≫3 + X≫5	Z = X≫1 + X≫3 + X≫5 + X≫6
	11	≤ 0.57	Z = X≫1	Z = X≫1 + X≫4	Z = X≫1 + X≫4 + X≫7	Z = X≫1 + X≫4 + X≫7 + X≫10
3	000	≤ 1	Z = X	Z = X	Z = X	Z = X
	001	≤ 0.88	Z = X	Z = X≫1 + X≫2	Z = X≫1 + X≫2 + X≫3	Z = X≫1 + X≫2 + X≫3 + X≫6
	010	≤ 0.80	Z = X	Z = X≫1 + X≫2	Z = X≫1 + X≫2 + X≫5	Z = X≫1 + X≫2 + X≫5 + X≫6
	011	≤ 0.72	Z = X≫1	Z = X≫1 + X≫2	Z = X≫1 + X≫3 + X≫4	Z = X≫1 + X≫3 + X≫4 + X≫5
	100	≤ 0.66	Z = X≫1	Z = X≫1 + X≫3	Z = X≫1 + X≫3 + X≫5	Z = X≫1 + X≫3 + X≫5 + X≫6
	101	≤ 0.61	Z = X≫1	Z = X≫1 + X≫4	Z = X≫1 + X≫4 + X≫5	Z = X≫1 + X≫4 + X≫5 + X≫6
	110	≤ 0.57	Z = X≫1	Z = X≫1 + X≫4	Z = X≫1 + X≫4 + X≫7	Z = X≫1 + X≫4 + X≫7 + X≫10
	111	≤ 0.53	Z = X≫1	Z = X≫1 + X≫5	Z = X≫1 + X≫5 + X≫9	Z = X≫1 + X≫5 + X≫9 + X≫13
4	0000	≤ 1	Z = X	Z = X	Z = X	Z = X
	0001	≤ 0.94	Z = X	Z = X	Z = X≫1 + X≫2 + X≫3	Z = X≫1 + X≫2 + X≫3 + X≫4
	0010	≤ 0.88	Z = X	Z = X≫1 + X≫2	Z = X≫1 + X≫2 + X≫3	Z = X≫1 + X≫2 + X≫3 + X≫6
	0011	≤ 0.84	Z = X	Z = X≫1 + X≫2	Z = X≫1 + X≫2 + X≫4	Z = X≫1 + X≫2 + X≫4 + X≫5
	0100	≤ 0.80	Z = X	Z = X≫1 + X≫2	Z = X≫1 + X≫2 + X≫5	Z = X≫1 + X≫2 + X≫5 + X≫6
	0101	≤ 0.76	Z = X≫1	Z = X≫1 + X≫2	Z = X≫1 + X≫2 + X≫7	Z = X≫1 + X≫2 + X≫7 + X≫11
	0110	≤ 0.72	Z = X≫1	Z = X≫1 + X≫2	Z = X≫1 + X≫3 + X≫4	Z = X≫1 + X≫3 + X≫4 + X≫5
	0111	≤ 0.69	Z = X≫1	Z = X≫1 + X≫3	Z = X≫1 + X≫3 + X≫4	Z = X≫1 + X≫3 + X≫4 + X≫7
	1000	≤ 0.66	Z = X≫1	Z = X≫1 + X≫3	Z = X≫1 + X≫3 + X≫4	Z = X≫1 + X≫3 + X≫5 + X≫7
	1001	≤ 0.64	Z = X≫1	Z = X≫1 + X≫3	Z = X≫1 + X≫3 + X≫7	Z = X≫1 + X≫3 + X≫7 + X≫11
	1010	≤ 0.61	Z = X≫1	Z = X≫1 + X≫3	Z = X≫1 + X≫4 + X≫5	Z = X≫1 + X≫4 + X≫5 + X≫6
	1011	≤ 0.59	Z = X≫1	Z = X≫1 + X≫4	Z = X≫1 + X≫4 + X≫6	Z = X≫1 + X≫4 + X≫6 + X≫7
	1100	≤ 0.57	Z = X≫1	Z = X≫1 + X≫4	Z = X≫1 + X≫4 + X≫7	Z = X≫1 + X≫4 + X≫7 + X≫10
	1101	≤ 0.55	Z = X≫1	Z = X≫1 + X≫5	Z = X≫1 + X≫5 + X≫6	Z = X≫1 + X≫5 + X≫6 + X≫8
	1110	≤ 0.53	Z = X≫1	Z = X≫1 + X≫5	Z = X≫1 + X≫5 + X≫9	Z = X≫1 + X≫5 + X≫9 + X≫13
	1111	≤ 0.51	Z = X≫1	Z = X≫1 + X≫6	Z = X≫1 + X≫6 + X≫12	Z = X≫1 + X≫6 + X≫12 + X≫13

Here p_1, p_2, ... p_n are integers and $\gg$ denotes right shift. The sum of shifted $X's$ is equal to the final value Z. In *FPAD* we introduce approximation in two stages. The first stage looks into of the number of bits of Y to be taken into consideration and the second stage maps the approximate value of the inverse of Y to shift add operations.

A. First Stage of Approximation

We first introduce approximation by limiting the number of bits of *mantissa* (Y) that will be used for the final computations. Limiting the number of bits will lead to a reduction in accuracy but will benefit in terms of power and delay. The most significant α bits of *mantissa* (Y) are selected. Here, "α" denotes the *level* of approximation. Based on the *levels*, the accuracy of the result can be tuned. With a given *level*, the approximate inverse of divisor *mantissa* (Y) are computed offline and are shown for α values of 1, 2, 3 and 4 in Table I.

B. Second Stage of Approximation

In this stage, we introduce approximation by further approximating (1/Y) value to the sum of powers of two. We find the value, closest to the approximate (1/Y) value obtained from stage 1, using only shifts and add. We come up with various FPAD designs by limiting the number of adders used to obtain the value closest to approx (1/Y). The number of adders decides the accuracy within a given level. As the number of adder units increases, the maximum error decreases.

Table I explains the approximation at different *levels* and also within a *level* by varying the number of adders. The nomenclature for our design is $\mathbf{L}\alpha\mathbf{A}\beta$ where α denotes the *level* and β denotes the number of adders.

TABLE II: Error values using for various levels with three adders

Approximation Level	Maximum Error(%)	Mean Error(%)
1	49.99	19.96
2	24.99	9.66
3	12.49	4.5
4	6.24	1.97

C. Error vs Level Trade-off

As error resilience varies across applications, a tailored *FPAD* can be used to achieve the desired accuracy. We used

10 million pairs of uniformly sampled floating point numbers as inputs to obtain the error distribution. The error result at different *levels* of approximation using 3 adders is shown in Table II. It can be seen that the maximum error is bounded for every level and also reduces as the number of *levels* increases.

D. Approximation Using Bit Truncation

Approximation can also be introduced in *mantissa* X. We can further increase approximation by truncating the bits of X for improving the area, power, and delay of the design. We observed that even after truncating the least significant 18-bits of X, the mean error of the proposed divider design for a given *level* of approximation does not degrade much. This can be attributed to the exponential reduction in the contribution of a bit towards the output value.

In conclusion, *FPAD* introduces approximation in multiple levels, and also within each level. Approximation can further be introduced by truncating the number of bits in mantissa X to further improve energy and performance without much reduction in output quality. Since *FPAD* is based on a multi-stage mathematical approximation methodology, we can tailor the divider depending upon the application's need. Fig. 1 shows the hardware implementation of *FPAD*. The *sign* bits are XORed, and the *exponent* of the divisor is subtracted from the *exponent* of the dividend, as shown in Fig. 1. Mantissa division is performed depending upon the required output quality by selecting a level and number of adders corresponding to that level.

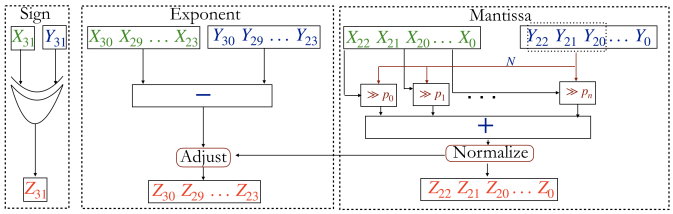

Fig. 1: Hardware implementation of *FPAD*

III. Evaluation and Results

We implemented all the FPAD designs in *Verilog* and synthesized the designs with *Synopsys Design Compiler* in *FDSOI 28 nm* technology. The power is calculated for all the designs at a constant frequency of 2 GHz.

Table III shows the error, area, power, delay and power delay product (PDP) result for *SP IEEE 754*, *CADE* ($N = 2$, $L = 4$) , *FaNZeD* and the proposed *FPAD*-t dividers. Here, the parameter 't' represents the number of truncated *mantissa*(X) bits. We have used *FloPoCo* [30] *SP IEEE 754* divider for our analysis. For the experiment, we have analyzed our two designs, *FPAD L4A3* and *FPAD L4A2*. The *FPAD L4A3-0* offers $682\times$ improvement in PDP as compared to *SP IEEE 754* divider. As compared to the *CADE* and *FaNZeD*, *FPAD L4A2-0* provides $12.36\times$ and $1.14\times$ reduction in PDP. Since CADE has a much higher PDP than *FaNZeD*, we have limited our further comparisons to *FaNZeD*. *FPAD L4A3-18* has a mean error of 2.88, similar to *FaNZeD* divider, but offers a $2.84\times$ improvement in PDP. It is evident from Table III that for a given mean error, *FPAD* provides higher energy efficiency as compared to *FaNZeD* dividers. Fig 2 compares *FaNZeD*, *FPAD L4A2*, *FPAD L4A3* and *FPAD L2A2* with truncation up to 18 bits for mean error vs. power, delay and PDP. It can be seen in Fig 2, that for a given PDP, *FPAD* designs are better as compared to *FaNZeD*.

TABLE III: Results for the proposed *FPAD* dividers

Divider	Mean Error(%)	Area (μm^2)	Power (μW)	Delay (ns)	PDP (fJ)
Existing Designs					
SP IEEE 754 [30]	-	9648.26	26727.4	0.31	8284.37
CADE (N=2, L=4) [15]	0.58	495.80	937.92	0.14	131.31
FaNZeD [17]	2.89	610.69	52.5	0.23	12.08
FaNZeD-20 [17]	3.81	208.73	19.65	0.17	3.24
Proposed Designs					
FPAD L4A3-0	1.97	1506.50	35.35	0.34	12.13
FPAD L4A3-18	2.88	290.01	19.89	0.25	4.26
FPAD L4A2-0	1.85	913.10	33.21	0.32	10.62
FPAD L4A2-18	4.61	323.95	17.18	0.22	3.78
FPAD L4A1-0	4.92	737.36	33.07	0.27	8.93
FPAD L4A1-18	6.32	266.51	17.00	0.18	3.05
FPAD L3A3-0	4.5	980.14	32.88	0.33	10.85
FPAD L3A3-18	3.44	274.01	16.62	0.23	3.82
FPAD L3A2-0	3.52	777.97	32.35	0.31	10.03
FPAD L3A2-18	3.78	273.64	16.60	0.22	3.65
FPAD L3A1-0	5.41	543.78	31.90	0.26	8.33
FPAD L3A1-18	6.15	252.31	16.59	0.17	2.86
FPAD L2A3-0	9.66	711.55	32.18	0.29	9.33
FPAD L2A3-18	6.55	270.10	16.27	0.16	2.60
FPAD L2A2-0	8.6	677.44	31.39	0.29	9.10
FPAD L2A2-18	6.55	270.09	16.26	0.16	2.6
FPAD L2A1-0	7.16	472.79	31.03	0.26	8.06
FPAD L2A1-18	6.45	256.06	16.25	0.16	2.6
FPAD L1A3-0	20.98	676.63	31.16	0.29	9.03
FPAD L1A3-18	17.68	223.42	15.70	0.15	2.35
FPAD L1A2-0	19.96	559.77	30.81	0.26	8.32
FPAD L1A2-18	17.68	223.42	15.70	0.15	2.35
FPAD L1A1-0	18.41	539.87	30.81	0.23	7.08
FPAD L1A1-18	16.98	223.42	15.70	0.15	2.35

IV. Applications

A. Image Processing Applications

Multimedia applications have been shown to benefit as a result of approximate computations [5]. We have used three applications: JPEG compression, mean filtering, and image enhancement for the evaluation of two standard 512×512 pixel grayscale images (cameraman and mandrill) [31].

We applied our approximate divider to JPEG Compression. It is the most common image compression standard for storing images in compressed format. The proposed dividers are employed in the quantization step. To evaluate the quality of the compressed image, we compared it with the original uncompressed image using Peak Signal to Noise Ratio (PSNR) for all the dividers. Table IV shows the results of JPEG compression on two standard 512×512 pixel grayscale images

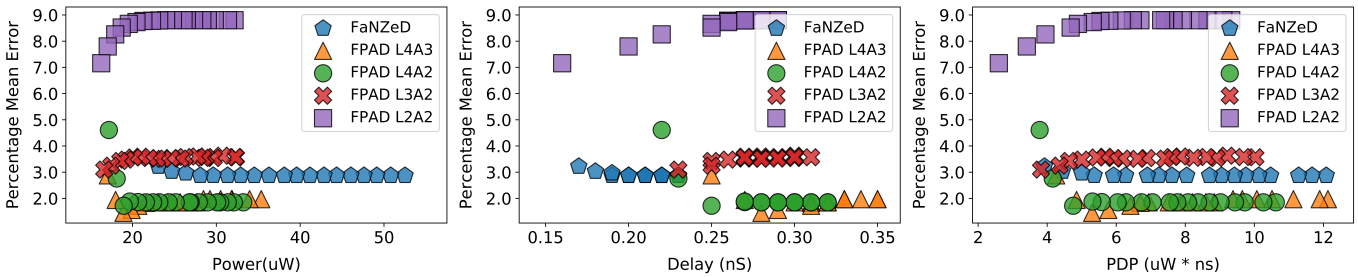

Fig. 2: Mean Error Vs Power, Delay and PDP for *FaNZeD*, *FPAD L4A3*, *FPAD L4A2*, *FPAD L3A2*, *FPAD L2A2*

TABLE IV: JPEG output quality results (in PSNR) using different approximate dividers

Divider	PDP (μW*ns)	Image:Cameraman Q:25	Q:50	Q:75	Image:Mandrill Q:25	Q:50	Q:75
IEEE sp FP	2171.70	32.6	35.7	38.6	24.3	26.2	28.7
FPAD L4A3	3.18	32.6	35.8	38.7	24.1	26.1	28.6
FPAD L4A2	2.78	32.3	35.3	38.3	24.1	26.1	28.5
FPAD L4A3-18	1.17	32.5	35.0	37.0	24.1	26.0	28.3
FPAD L4A2-18	0.99	32.2	34.7	36.8	24.1	26.0	28.3
FPAD L3A2-18	0.96	32.3	34.8	36.8	24.2	26.0	28.3
***FPAD* L2A2-18**	0.68	**31.8**	**32.9**	**35.6**	**24.2**	**26.3**	**28.5**
FPAD L1A2-18	0.62	30.0	30.1	29.5	23.9	24.9	25.7

(a) IEEE 754; PSNR = 38.6 (b) L1A3-18; PSNR = 29.5 (c) L2A2-18; PSNR = 35.6 (d) L3A1-18; PSNR = 36.2

(e) IEEE 754; PSNR = 28.7 (f) L1A3-18; PSNR = 25.7 (g) L2A2-18; PSNR = 28.5 (h) L3A1-18; PSNR = 28.0

Fig. 3: Output quality using *FPAD* for JPEG compression

at quality Level 25, 50, and 75, respectively. We also show some of the output images in Fig. 3.

Mean filtering is used to smoothen the images. We used *FPAD L2A2-18* and obtained a PSNR value of 47.9 and 41.8 for cameraman and mandril, respectively. In image enhancement process consists of Gaussian blur filtering to denoise the image, followed by a Laplace filter to detect the edges. Both images are added back to the original image. We used *FPAD L2A2-18* in the Gaussian filter step and obtained a PSNR of 40.7 and 40.4 for cameraman and mandrill, respectively.

B. Convolutional Neural Network

Convolutional neural networks have been successfully applied to several image processing and computer vision tasks like image classification, object detection, etc [32]. We have used PyTorch implementation of AlexNet [33] for the experiments. However, to replace the exact divider by the proposed floating point divider in local response normalization(LRN) layer and softmax layer, we implemented a custom LRN and softmax layer in *python*. We selected 5000 images from the validation set of ImageNet [34] dataset for evaluation. We have used the pre-trained weights of AlexNet trained on Imagenet training images. In AlexNet, there are nearly 0.5 million division operations. We have reported the top-1, and top-5 errors as these are widely used metrics to evaluate the performance of CNNs. Table V shows that *FPAD L2A2-18* has less than 2% increase in top-1 error and $\sim 1\%$ increase in top-5 error while offering 25% PDP improvement over *FaNZeD*-20.

TABLE V: AlexNet classification accuracy and energy using various approximate dividers

Divider	SP IEEE 754	CADE	FaNZeD-20	FPAD L4A3	FPAD L4A3-18	FPAD L2A2-18
PDP (μW*ns)	3951.8	13.131	1.56	5.78	2.03	1.24
Top-1	43.00	43.00	43.20	43.64	43.72	43.76
Top-5	19.20	19.30	20.00	20.38	20.30	20.36

V. Conclusion

In this paper, we propose *FPAD*, a novel approximate floating point divider based on a simple mathematical approach of breaking down the *mantissa* division into a series of shifts and addition operations. We also provide a wide range of *FPAD* designs with a varying tradeoff in error and PDP. The proposed divider bounds the maximum error at every *level* of approximation. At each level, the number of adders can be changed for a different area, power, and delay improvement. We also used *FPAD* designs in image processing applications and to perform inferencing on Alexnet CNN to show the tradeoff between output quality and PDP.

Acknowlegements

This work was supported by Intel India PhD Fellowship and MeitY, India under Grant Visvesvaraya PhD scheme and SMDP-C2SD.

REFERENCES

[1] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS)*. IEEE, 2013, pp. 1–6.

[2] B. Moons, D. Bankman, and M. Verhelst, "Circuit techniques for approximate computing," in *Embedded Deep Learning*. Springer, 2019, pp. 89–113.

[3] D. Ray, N. V. George, and P. K. Meher, "Analysis and design of approximate inner-product architectures based on distributed arithmetic," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019, pp. 1–5.

[4] C. K. Jha, S. N. Ved, I. Anand, and J. Mekie, "Energy and error analysis framework for approximate computing in mobile applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2019.

[5] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, p. 62, 2016.

[6] D. Esposito, A. G. Strollo, and M. Alioto, "Power-precision scalable latch memories," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[7] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, "Cross-layer approximate computing: From logic to architectures," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[8] R. Boyapati, J. Huang, P. Majumder, K. H. Yum, and E. J. Kim, "Approx-noc: A data approximation framework for network-on-chip architectures," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 666–677.

[9] A. Sampson, A. Baixo, B. Ransford, T. Moreau, J. Yip, L. Ceze, and M. Oskin, "Accept: A programmer-guided compiler framework for practical approximate computing," *University of Washington Technical Report UW-CSE-15-01*, vol. 1, no. 2, 2015.

[10] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "Enerj: Approximate data types for safe and general low-power computation," *ACM SIGPLAN Notices*, vol. 46, no. 6, pp. 164–174, 2011.

[11] X. Yi, H. Pei, Z. Zhang, H. Zhou, and Y. He, "Design of an energy-efficient approximate compressor for error-resilient multiplications," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019, pp. 1–5.

[12] E. Adams, S. Venkatachalam, and S.-B. Ko, "Energy-efficient approximate mac unit," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019, pp. 1–4.

[13] S. Venkatachalam, E. Adams, and S.-B. Ko, "Design of approximate restoring dividers," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019, pp. 1–5.

[14] M. K. Jaiswal and H. K.-H. So, "Area-efficient architecture for dual-mode double precision floating point division," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 2, pp. 386–398, 2016.

[15] M. Imani, R. Garcia, A. Huang, and T. Rosing, "Cade: Configurable approximate divider for energy efficiency," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 586–589.

[16] S. Hashemi, R. I. Bahar, and S. Reda, "A low-power dynamic divider for approximate applications," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: ACM, 2016, pp. 105:1–105:6. [Online]. Available: http://doi.acm.org/10.1145/2897937.2897965

[17] H. Saadat, H. Javaid, and S. Parameswaran, "Approximate integer and floating-point dividers with near-zero error bias," in *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 2019, p. 161.

[18] T. Zhang, W. Liu, E. McLarnon, M. O'Neill, and F. Lombardi, "Design of majority logic (ml) based approximate full adders," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[19] T. Yang, T. Ukezono, and T. Sato, "A low-power yet high-speed configurable adder for approximate computing," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[20] W. Liu, J. Xu, D. Wang, C. Wang, P. Montuschi, and F. Lombardi, "Design and evaluation of approximate logarithmic multipliers for low power error-tolerant applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 9, pp. 2856–2868, 2018.

[21] S. Venkatachalam, H. J. Lee, and S.-B. Ko, "Power efficient approximate booth multiplier," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–4.

[22] D. Esposito, D. De Caro, E. Napoli, N. Petra, and A. G. Strollo, "On the use of approximate adders in carry-save multiplier-accumulators," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[23] C. K. Jha and J. Mekie, "Seda-single exact dual approximate adders for approximate processors," in *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 2019, p. 237.

[24] S. Vahdat, M. Kamal, A. Afzali-Kusha, M. Pedram, and Z. Navabi, "Truncapp: A truncation-based approximate divider for energy efficient dsp applications," in *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2017, pp. 1639–1642.

[25] M. Imani, Y. Kim, A. Rahimi, and T. Rosing, "Acam: Approximate computing based on adaptive associative memory with online learning," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*. ACM, 2016, pp. 162–167.

[26] M. Imani, S. Gupta, and T. Rosing, "Ultra-efficient processing in-memory for data intensive applications," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 2017, p. 6.

[27] S. Jain, M. Pancholi, H. Garg, and S. Saini, "Binary division algorithm and high speed deconvolution algorithm (based on ancient indian vedic mathematics)," in *2014 11th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*. IEEE, 2014, pp. 1–5.

[28] C. Jha and J. Mekie, "Design of novel cmos based inexact subtractors and dividers for approximate computing: An in-depth comparison with ptl based designs," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*, Aug 2019, pp. 174–181.

[29] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, July 2019.

[30] F. de Dinechin, J. Detrey, O. Creţ, and R. Tudoran, "When FPGAs are better at floating-point than microprocessors," Sep. 2007, working paper or preprint. [Online]. Available: https://hal-ens-lyon.archives-ouvertes.fr/ensl-00174627

[31] [Online]. Available: https://homepages.cae.wisc.edu/ ece533/images/

[32] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, Dec 2015. [Online]. Available: https://doi.org/10.1007/s11263-015-0816-y

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[34] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.