FPCAM: Floating Point Configurable Approximate Multiplier for Error Resilient Applications

Chandan Kumar Jha, Sumit Walia, Gagan Kanojia, and Joycee Mekie
Electrical Engineering Department
Indian Institute of Technology Gandhinagar, Gandhinagar, India
Email: {chandan.jha, sumit.walia, gagan.kanojia, and joycee}@iitgn.ac.in

2021 IEEE International Symposium on Circuits and Systems (ISCAS) | 978-1-7281-9201-7/20/$31.00 ©2021 IEEE | DOI: 10.1109/ISCAS51556.2021.9401341

***Abstract*—In this paper, we propose the design of a power-efficient floating point configurable approximate multiplier (FP-CAM) suitable for error resilient applications. FPCAM allows systematic approximation, and the amount of approximation can be configured at run-time, depending on the error-tolerance of the applications. We show that compared to the existing state of the art multipliers, FPCAM on average consumes 62% lesser power and has 69% less power delay product. FPCAM also has 66% less area as compared to state of the art approximate multipliers. We have analyzed FPCAM for three different multimedia applications and random inputs to show that we achieve similar output quality as compared to existing multipliers while benefiting in power, area, and power delay product.**

I. Introduction

Approximate circuits have been gaining a lot of popularity in recent years [1], [2]. Approximate circuits exploit the application's resilience towards the introduction of approximation to provide benefits in power, area, and delay [3]–[5]. The benefits of using approximate circuits have been shown on image and video processing applications in many earlier works using integer arithmetic computation [6]–[10]. It has also been shown in recent works that floating point arithmetic units benefit from the introduction of approximation [11]–[17]. In this paper, we will focus on the design of a configurable approximate floating point (FP) multiplier [14]–[17].

FP multipliers perform multiplication of two floating point numbers. Floating point numbers are represented using the IEEE 754 format. The IEEE 754 32-bit representation is as shown in Fig. 1. Some of the previous configurable approximate floating point multiplier designs have both approximate and exact multiplication units, and the decision to perform approximate or exact multiplication is data-dependent [14], [15]. In CFPU [14], the consecutive most significant mantissa bits called the tuning bits, are checked to see if all are either 1's or 0's. If all the tuning bits are 1's or 0's, the exponent is increased by one or kept the same respectively. In case all tuning bits are not the same an exact multiplication is done. In RMAC [15], the mantissa bits of the inputs are added. If the most significant bit of both the input mantissa's are 1 or 0, consecutive bits in the resultant mantissa are checked for all 1's and 0's respectively. The consecutive bits checked are called the tuning bits similar to CFPU. If the tuning bits are the same the added result is made the output, else exact multiplication is performed. In ApproxLP [16], unlike CFPU and RMAC, no exact multiplier is used. It uses an

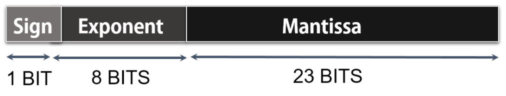


Fig. 1. IEEE 754 32-bit Representation

iterative methodology to performed weighted additions. As the number of iterations increases, called levels in ApproxLP, the error reduces. Unfortunately, with the increase in levels, the hardware complexity increases exponentially as the number of comparisons performed increases and different weighted additions are performed. In [17], the FP multiplier that has a minimum bias in error is proposed. This design suffers from the limitation of not being configurable and uses an approximate multiplier to perform the mantissa multiplication.

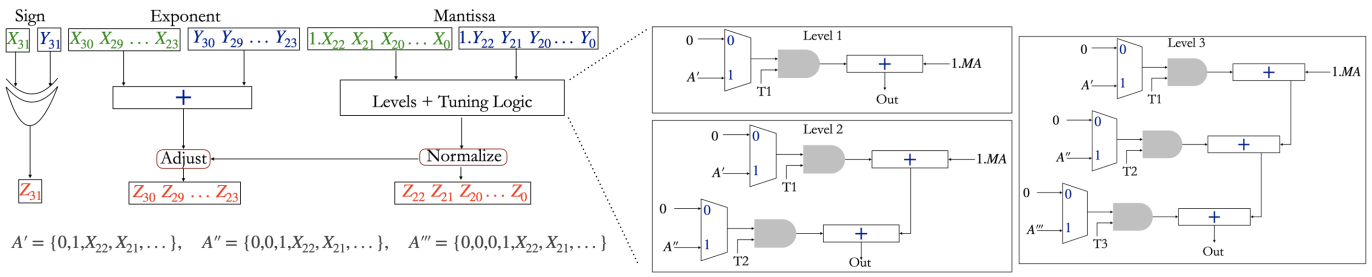
In this paper, we propose a scalable, energy-efficient, dynamically-configurable approximate floating point (FP) multiplier design based on a systematic mathematical approximation approach. The proposed floating point configurable approximate multiplier (FPCAM) overcomes the limitations of the prior works [14]–[17]. FPCAM does not need a separate exact unit, unlike in [14], [15]. Since FPCAM is based on a mathematical approach, the hardware increases linearly with the increase in levels unlike [16]. FPCAM also supports runtime configurability as it uses only shift and addition operations. In this work, we have compared FPCAM with the existing configurable FP multipliers [14], [15], and the designs which allow multiple levels of approximation [16]. Our contributions are as follows.

- We proposed the design of a floating point configurable approximate multiplier (FPCAM) which unlike all existing designs does not require the implementation of an exact multiplier and scales linearly with increase in levels.
- The proposed design allows user run time configurability depending upon the amount of approximation required.
- We show that FPCAM gives power, area, and power delay product benefits as compared to the state of the art when used in multimedia applications.

II. FPCAM

A. Mathematical Analysis for Approximation

Let us assume we have two numbers A and B in IEEE 754 format. Let us represent their mantissas as MA and MB



Sign
Exponent
Mantissa
Levels + Tuning Logic
Adjust
Normalize
$A' = \{0,1,X_{22},X_{21},\dots\}$, $A'' = \{0,0,1,X_{22},X_{21},\dots\}$, $A''' = \{0,0,0,1,X_{22},X_{21},\dots\}$
Level 1
Level 2
Level 3
T1
T2
T3
Out
1.MA

Fig. 2. Proposed Floating point configurable approximate multiplier (FPCAM)

respectively.

$$MA = 1.X_{22}X_{21}X_{20}\dots x_0, MB = 1.Y_{22}Y_{21}Y_{20}\dots Y_0 \quad (1)$$

The numeric value of the mantissa for Equation 1 can be obtained as,

$$MA = 1 + 0.5 * X_{22} + 0.25 * X_{21} + 0.125 * X_{20} + \dots$$
$$MB = 1 + 0.5 * Y_{22} + 0.25 * Y_{21} + 0.125 * Y_{20} + \dots$$

If we have to perform exact multiplication we would multiply MA and MB to obtain the result. The expression would look like

$$MA * MB = MA * (1 + 0.5 * Y_{22} + 0.25 * Y_{21} + \dots)$$
$$= 1 * MA + 0.5 * Y_{22} * MA + 0.25 * Y_{21} * MA + \dots \quad (2)$$

We rearrange Equation 2 to obtain the following

$$MA * MB = (MA) + (0.5 * MA) * Y_{22}$$
$$+ (0.25 * MA) * Y_{21} + (0.125 * MA) * Y_{20} \dots \quad (3)$$

The design of FPCAM is shown in Fig. 2. The sign bit and the exponent generation are done using the same method used in an exact multiplier. The sign bits are XORed to obtain the resultant sign bit and the exponents are added to get the final exponent of the product. The hardware for mantissa multiplication is built using equation 3. The equation has been rearranged to benefit in designing the hardware. From equation 3, it can be seen MA, 0.5*MA, 0.25*MA, and so on can be obtained by selecting the input MA and appending $0's$. An important point to note is that this does not lead to an increase in the hardware.

The second set of multiplications are the product of MA, 0.5*MA, 0.25*MA, and so on, with individual bits of MB's, respectively. From equation 3, we see that it is the same as deciding whether or not we need to include a shifted version of MA in the final expression, i.e. if b23 is 0, the term multiplied to it $(0.5 * MA)$ will give the result 0 i.e. it will have no contribution to the final output. Similarly, if b23 is 1, the term multiplied to it $(0.5*MA)$ will give the result $(0.5*MA)$ which needs to be added to the other terms for which the entries of MB's the mantissa have a non zero value. We propose to use the bits of mantissa of MB as a control signal to MUX that will pass the value to be added. If the control signal to the MUX is 0, the output of the MUX will be 0. If the control signal of the MUX is 1, the output will be the same as the input. Thus we see that the logic inside the MUX will reduce to an AND gate. Depending upon the required accuracy for a particular application we can choose how many addition terms need to be included. Each addition term forms a layer, which consists of a MUX, AND, and an adder as shown in Fig. 2. To maintain consistency in terminology we will use the term, *levels*, to decide the number of layers required for approximation. We will perform the addition of the individual results using adders which gives us the final approximate answer. If we generate a carry in the final result we increase the exponent by 1, else the exponent is the same.

Within each level, the tuning bits T1, T2, and T3 can be used to further increase the approximation. We have introduced AND gates in the design that act as masks. These AND gates will suppress the output of layers when higher accuracy is not required and will give benefits as shown in Section III-C. For example, if we select a level-3 FPCAM, it will have three layers as shown in Fig. 2. If we want only one layer operation, we can mask the input of layer 2 and 3 by setting T1 to 1, T2, and T3 to 0. If two layer operation is needed, we can mask the input by setting T1 and T2 to 1 and T3 to 0. Lastly, if all the levels are needed we can make T1, T2, and T3 to be all 1's. Thus, the applications that have the least error tolerance will decide the number of layers in FPCAM. Further, if that same design needs to be used in an application that can tolerate more errors, the tuning bits can be used to reduce the number of levels at runtime.

Thus, FPCAM allows the designer to select the number of layers in an optimal way rather than wasting resources. It can be scaled linearly depending upon the error requirements, and further can be configured at runtime to trade-off output quality and power and power delay product.

III. Evaluation and Results

The designs were implemented using the Verilog language. All the designs were synthesized using Synopsys Design Compiler (DC). We have used 28nm FDSOI technology node for synthesis. The operating condition was for 1V, 25° C, and typical corner. The designs were synthesized for the maximum frequency to obtain the technology dependent netlist, area, and delay values. The switching activity file was generated using Synopsys VCS, and a frequency of 1.6 GHz was used for all designs to obtain power using the Synopsys DC tool. Python scripts were used to obtain error values. The applications

TABLE I
POWER COMPARISON OF VARIOUS APPROXIMATE MULTIPLIERS

		Power (uW)			
Applications	Tune/ Levels	**FPCAM**	CFPU [14]	RMAC [15]	ApproxLP [16]
BLUR	1	91.41	344.42	453.61	158.76
	2	120.59	446.66	388.03	261.44
	3	151.89	500.54	368.61	420.45
	4	151.81	586.61	356.50	-
	Average	128.92	469.56	391.69	280.21
DWT	1	84.79	191.09	237.66	117.86
	2	97.95	229.79	219.12	177.39
	3	116.68	256.43	209.30	275.58
	4	117.41	347.64	205.31	-
	Average	104.21	256.24	217.85	190.28
DCT	1	143.00	575.41	740.21	198.25
	2	183.40	710.70	685.03	351.65
	3	213.38	775.75	642.32	564.97
	4	221.57	851.82	598.83	-
	Average	190.34	728.42	666.60	371.62
RANDOM	1	141.78	579.23	766.03	220.25
	2	198.87	727.55	663.48	387.13
	3	241.76	829.88	634.07	618.32
	4	254.04	907.89	613.59	-
	Average	209.11	761.14	669.29	408.57

TABLE II
PERCENTAGE OF EXACT COMPUTATIONS IN CFPU AND RMAC

	Tune	DWT	DCT	BLUR	RANDOM
CFPU	1	10.6	56.7	75.0	49.0
	2	3.6	44.8	60.0	24.0
	3	1.2	31.6	53.0	11.0
	4	0.4	23.3	41.0	5.0
RMAC	1	12.0	30.0	23.0	24.0
	2	9.0	29.5	3.0	18.0
	3	16.0	37.0	21.0	29.0
	4	22.0	40.0	33.0	39.0

used are Discrete Wavelet Transform (DWT), Discrete Cosine Transform (DCT), and BLUR applied on a 256×256 Lena image. We have also evaluated the same on RANDOM, where we generated one million random inputs sampled from a uniform distribution using Python NumPy library [18].

A. Power, Delay, PDP and Area Comparison

We first discuss the power consumption as shown in Table I. The power consumption of CFPU and RMAC is very high. This happens because these designs have both exact and approximate multipliers. Since depending upon the input data either approximate or exact computations are done, the percentage of exact computations causes these designs to consume a lot of power. The percentage of exact computations for varying tuning bits and applications are shown in Table II. In ApproxLP, we see that as the levels increase the power consumption increases. In all cases, we see that FPCAM consumes the least power for all levels. On average across all applications, FPCAM consumes 71%, 67%, and 49% lesser power as compared to CFPU, RMAC, and ApproxLP respectively.

In Table III we have shown the delay values of all the designs. We see that CFPU and RMAC have the highest delays

TABLE III
DELAY COMPARISON OF VARIOUS APPROXIMATE MULTIPLIERS

	Delay (ns)			
Tune/Levels	**FPCAM**	CFPU [14]	RMAC [15]	ApproxLP [16]
1	0.21	0.55	0.55	0.17
2	0.25	0.55	0.55	0.22
3	0.29	0.55	0.55	0.25
4	0.33	0.55	0.55	-

TABLE IV
PDP COMPARISON OF VARIOUS APPROXIMATE MULTIPLIERS

		PDP(μW*ns)			
Applications	Tune/ Levels	**FPCAM**	CFPU [14]	RMAC [15]	ApproxLP [16]
BLUR	1	19.20	189.43	249.49	26.99
	2	30.15	245.67	213.42	57.52
	3	44.05	275.30	206.42	105.11
	4	50.10	328.50	199.64	-
	Average	35.87	259.72	217.24	63.21
DWT	1	17.81	105.10	130.71	20.04
	2	24.49	126.38	120.51	39.03
	3	33.84	141.04	117.21	68.89
	4	38.75	194.68	114.97	-
	Average	28.72	141.80	120.85	42.65
DCT	1	30.03	316.47	407.11	33.70
	2	45.85	390.88	376.77	77.36
	3	61.88	426.66	359.70	141.24
	4	73.12	477.02	335.34	-
	Average	52.72	402.76	369.73	84.10
RANDOM	1	29.77	318.58	421.31	37.44
	2	49.72	400.15	364.92	85.17
	3	70.11	456.44	355.08	154.58
	4	83.83	508.42	343.61	-
	Average	58.36	420.90	371.23	92.40

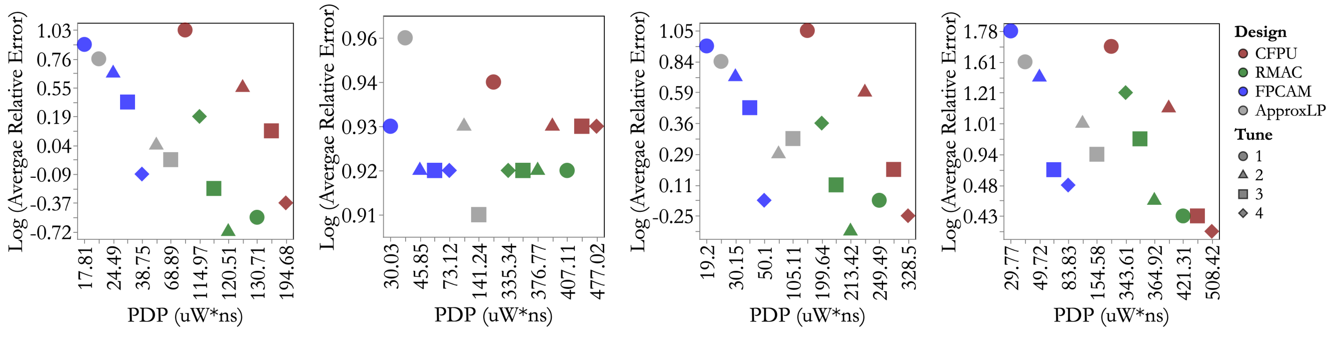
and irrespective of the tuning bits the delay for CFPU and RMAC is the same as it is decided by the exact floating point multiplier unit. We see that ApproxLP has the least delay among all the multipliers. For the same level, we see that FPCAM has a delay overhead of 25%, 13%, and 16% for levels 1, 2, and 3 respectively. While FPCAM has higher delay values but there is an overall benefit in the power delay product (PDP) as shown in Table IV. We see that FPCAM has the least PDP for all applications for various levels. On average across all applications, FPCAM has 85%, 83%, and 38% lesser PDP as compared to CFPU, RMAC, and ApproxLP respectively.

In Table V we have shown the area values of all the designs. We see that CFPU and RMAC have a large area. This is again due to the use of the exact multiplier unit. FPCAM has the least area among all the designs. On average FPCAM has 75%, 75%, and 46% lesser area as compared to CFPU, RMAC, and ApproxLP respectively.

Overall we see that FPCAM is better is terms of all power, PDP and area as compared to the state of the art approximate multiplier designs.

B. Error vs PDP Plots for Various Applications

In this work, we have compared FPCAM using three applications and one random input stream. We have computed the



(a) DWT (b) DCT (c) BLUR (d) RANDOM

Fig. 3. Log(Average Relative Error) vs PDP for various applciations

TABLE V
AREA COMPARISON OF VARIOUS APPROXIMATE MULTIPLIERS

	Area (μm^2)			
Tune/Levels	**FPCAM**	CFPU [14]	RMAC [15]	ApproxLP [16]
1	435.42	3091.99	3051.84	645.46
2	736.85	3163.63	3220.43	1325.84
3	892.38	3166.57	3211.78	2313.03
4	998.95	3244.42	3166.73	-

error by comparing the outputs of the individual approximate multiplications to obtain the average relative error given by 4.

$$Average\ Relative\ Error = \frac{1}{N}\sum |exact - approximate| \tag{4}$$

where N is the total number of multiplications. Since the error values vary widely across applications we have taken the logarithm of the average relative error to make the plots readable. We have shown the plot for the logarithm of average relative error as compared to PDP in Fig. 3. The designs that have the least PDP for a given error are the FPCAM designs. While the designs CFPU and RMAC have the least error, the PDP values are quite large. We want to highlight that these applications are amenable to the introduction of approximation, i.e., we can tolerate large values of errors. For a given value of output error, FPCAM designs offer a wide range of choices with the least PDP.

C. PDP Comparison of Configurability in FPCAM

In Section II, we have shown that after a level is selected the FPCAM can be configured at runtime to reduce the power consumption. For example, once we select a level, say level FPCAM-4, we can configure it to reduce power consumption. We can configure FPCAM-4, to run in level 1, 2, 3, and 4 using the tuning bits. Similarly, FPCAM-3 can be configured to run in levels 1, 2, and 3. We have shown the PDP values of the design for all in Table VI. We want to highlight that when FPCAM-4 is used in level 3 it has a higher PDP as compared to FPCAM-3. This happens because there is still some additional power consumption in the levels where one input has been made 0 using the AND gate and also the delay is more. Hence,

TABLE VI
PDP OF VARIOUS FPCAM DESIGNS

Design-Levels	Tune Bits	BLUR	DWT	DCT	RANDOM
FPCAM-4	4	50.10	38.75	73.12	83.83
	3	48.76	38.60	71.60	81.14
	2	42.09	34.97	64.50	70.26
	1	32.71	30.26	51.86	50.90
FPCAM-3	3	42.27	33.84	61.88	70.11
	2	36.00	28.90	54.30	59.99
	1	23.82	22.00	37.08	37.78
FPCAM-2	2	30.15	24.49	45.85	49.72
	1	23.31	21.53	36.04	36.11
FPCAM-1	1	19.20	17.81	30.03	29.77

for an application, depending upon the maximum error and the required runtime configurability, FPCAM provides various design options.

IV. CONCLUSION

In this paper we proposed the design of a floating point configurable approximate multiplier unit called FPCAM. FPCAM does not use any exact multiplication units like CFPU [14] and RMAC [15]. FPCAM hardware also increases linearly with the increase in the number of levels and not exponentially like in ApproxLP [16]. Depending upon the maximum error, various versions of FPCAM can be selected and within each level the error can be configured at runtime depending upon the application's requirement. We analyzed FPCAM for image processing applications like BLUR, FFT and DWT. On average FPCAM consumes 62% lesser power and has 69% lesser power delay product as compared to the state of the art approximate floating point multipliers.

ACKNOWLEDGEMENTS

This work is supported through grants received from Intel, SMDP-C2SD and YFRF/PhD fellowship under Visvesvaraya PhD scheme from the Ministry of Electronics and IT, through SERB grants CRG/2018/005013 and MTR/2019/001605, and through SRC Grant 2980.001.

REFERENCES

[1] I. Scarabottolo, G. Ansaloni, G. A. Constantinides, L. Pozzi, and S. Reda, "Approximate logic synthesis: A survey," *Proceedings of the IEEE*, 2020.

[2] W. Liu, F. Lombardi, and M. Shulte, “A retrospective and prospective view of approximate computing,” *Proceedings of the IEEE*, vol. 108, no. 3, pp. 394–399, 2020.

[3] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, “Approximate arithmetic circuits: A survey, characterization, and recent applications,” *Proceedings of the IEEE*, 2020.

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

[5] H. Jiang, C. Liu, F. Lombardi, and J. Han, “Low-power approximate unsigned multipliers with configurable error recovery,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, no. 99, pp. 1–14, 2018.

[6] C. K. Jha, A. Nandi, and J. Mekie, “Quality tunable approximate adder for low energy image processing applications,” in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 642–645, IEEE, 2019.

[7] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, “Low-power digital signal processing using approximate adders,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124–137, 2012.

[8] A. G. M. Strollo, E. Napoli, D. De Caro, N. Petra, and G. Di Meo, “Comparison and extension of approximate 4-2 compressors for low-power approximate multipliers,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2020.

[9] 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)*, pp. 174–181, IEEE, 2019.

[10] A. Nandi, C. K. Jha, and J. Mekie, “Tunable inexact subtracters for division in image processing applications,” in *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1100–1103, IEEE, 2020.

[11] 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)*, pp. 586–589, IEEE, 2019.

[12] H. Saadat, H. Javaid, and S. Parameswaran, “Approximate integer and floating-point dividers with near-zero error bias,” in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2019.

[13] C. K. Jha, K. Prasad, V. K. Srivastava, and J. Mekie, “Fpad: a multistage approximation methodology for designing floating point approximate dividers,” in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2020.

[14] M. Imani, D. Peroni, and T. Rosing, “Cfpu: Configurable floating point multiplier for energy-efficient computing,” in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2017.

[15] M. Imani, R. Garcia, S. Gupta, and T. Rosing, “Rmac: Runtime configurable floating point multiplier for approximate computing,” in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 1–6, 2018.

[16] M. Imani, A. Sokolova, R. Garcia, A. Huang, F. Wu, B. Aksanli, and T. Rosing, “Approxlp: Approximate multiplication with linearization and iterative error control,” in *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

[17] H. Saadat, H. Bokhari, and S. Parameswaran, “Minimally biased multipliers for approximate integer and floating-point multiplication,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2623–2635, 2018.

[18] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, *et al.*, “Array programming with numpy,” *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.