ICCAD: U: Optimally Approximated Floating-Point Multiplier

Chuangtao Chen (Author), Cheng Zhuo (Research Advisor)
College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China
E-mail: chtchen@zju.edu.cn; ACM ID: **2321450**; Category: **Undergraduate**

I. PROBLEM AND MOTIVATION

At the edge, IoT devices are designed to consume the minimum resource to achieve the desired accuracy. However, the conventional processors, such as CPU or GPU, can only conduct all the computations with predetermined but sometimes unnecessary precisions, inevitably degrading their energy efficiency. When running data-intensive applications, due to the large range of input operands, most conventional processors heavily rely on floating-point units (FPUs). Recently, approximate computing has become a promising alternative to improve energy efficiency for IoT devices on the edge, especially when running inaccuracy-tolerable applications. For various data-intensive tasks on edge devices, multiplication is a common but the most energy consuming one among different floating-point operations. As a common arithmetic component that has been studied for decades [1]–[3], the past focus on the FP multiplier is accuracy and performance. It is then observed that significant energy and time are spent on FP multipliers computing highly accurate outputs that are not necessarily demanded for various inaccuracy-tolerable tasks.

Recently, with awareness of the compromise between the stringent resource constraint and the accuracy tolerance for edge tasks, researchers have growing interests in designing an approximate FP multiplier to improve energy efficiency [4]–[6]. **However, how to achieve an optimal approximation while guarantee unbiased error distribution remains an open question.** This is not a trivial task: (1) Unlike the many approximations in prior work that stem from heuristic findings [4]–[7], it is desired to formally define the problem, including objective function and constraints, to enable the theoretically sound basis for optimal approximation. (2) In addition to optimality, the approximate FP multiplier needs to be unbiased to support consecutive multiplications commonly met in edge tasks, which is also not straightforward. (3) Last but not the least, the underlying architecture should facilitate the circuitry implementation to prevent exponentially growing area complexity for higher precision requirements. In this work, we proposed an optimally approximated FP multiplier to address the aforementioned challenges. With the support of a theoretically sound formulation that turns multiplication approximation to an optimization problem, a multi-level architecture is proposed to easily incorporate unbiasedness, runtime configurability and module execution parallelism. To reduce the design complexity, an optimization scheme is applied, making the proposal linearly (instead of quadratically or exponentially) dependent on the precision. The efficiency and advantages of the proposed design has then been demonstrated by extensive experiments.

II. BACKGROUND AND RELATED WORK

Many prior designs on approximate multiplier attempt to tackle the problem either from gate or algorithmic levels to reduce the product bit-width or critical path delay. For example, some work uses approximate components, such as compressors, to build the multiplier, so as to speed up addition or partial product generation [7]–[12]. [7] leverages a modified 2×2 block to construct the inaccurate multiplier. However, the error is hard to control and unbiased error distribution cannot be guaranteed due to its gate level design. On the other hand, configurability is highly demanded for versatile edge scenarios. To address these issues, some work try to approximate multiplication from a higher design level. In [13], researchers propose to truncate the bits after the leading one to conserve energy and replace the truncated part with a "1" to have unbiased error distribution. [4], [5] utilize hybrid methods with both inaccurate and accurate multipliers to adjust the computational accuracy, thereby causing significant area consumption. Recently, ApproxLP is proposed to approximate the mantissa product using linear fitting [6]. The design has much higher performance for the given error rate when compared to the prior solutions, but also suffers from non-trivial area overhead and biased error distribution.

III. APPROACH AND UNIQUENESS

A. Problem Formulation

To achieve optimal approximation, we need to formally formulate the problem which can incorporate specified design targets. For the multiplication of two normalized FP numbers, the target function can be defined as:

$$f(x, y) = xy, \tag{1}$$

where x and y are two mantissas within the range of $[1, 2)$. Our idea is to project the complex multiplication function to another space V with lower dimension. Specifically, we select a group of bases $1, x, y, x^2, y^2$ to decouple the two inputs. We then can define:

$$V = span(1, x, y, x^2, y^2). \tag{2}$$

Given two continuous functions $g(x,y)$ and $h(x,y)$ in the domain of $[x_1,x_2]\times[y_1,y_2]$, we can define their inner product as:

$$< g,h >= \int_{x_1}^{x_2}\int_{y_1}^{y_2} g\times h\ dx\ dy. \tag{3}$$

It can be easily proved that the selected five bases are linearly independent when $x_2 > x_1 \geq 0$ and $y_2 > y_1 \geq 0$. It is obvious that $f \notin V$, so that an approximate projection of the multiplication f to the space V is necessary. As vectors in V are the linear combinations of the selected bases, we can define the following approximate function:

$$f_{approx} = k_0 + k_1 x + k_2 y + k_3 x^2 + k_4 y^2. \tag{4}$$

Without loss of generality, we can choose *square distance* as a mathematically friendly error measurement within the domain $[x_1,x_2]\times[y_1,y_2]$:

$$||f - f_{approx}||^2 =< f - f_{approx}, f - f_{approx} > . \tag{5}$$

To achieve the approximation with the minimal deviation from the original function, we can minimize the square error in Eq. (5), and obtain the following solution for k_i's:

$$\begin{aligned}&[k_0^*,k_1^*,k_2^*,k_3^*,k_4^*]\\ &=\left[-\frac{(x_1+x_2)(y_1+y_2)}{4},\frac{y_1+y_2}{2},\frac{x_1+x_2}{2},0,0\right].\end{aligned} \tag{6}$$

Note that the formulation above is not limited to the square error and these selected bases, but applicable to other different error measures and bases. Since the coefficients of the square items are both zero, the optimal approximation can be re-expressed as:

$$f = xy \approx f^*_{approx} = k_0^* + k_1^* x + k_2^* y. \tag{7}$$

For the error between f and f^*_{approx}, it can be found that the error distribution by its nature is symmetric around zero, making the error unbiased. In addition, as the square error is proportion to the cube of the domain area $(x_1-x_2)(y_1-y_2)$, which implies the total error can be further reduced by partitioning the domain into smaller sub-domains. Given the specified number of sub-domains, it can be proved that the total error is minimized when and only when the area of each sub-domain is equal.

B. Architecture for the Proposed Multiplier

Based on those theoretical findings, we design a multi-level architecture for the proposed FP multiplier as in Fig. 1. Level 0, as the basic approximation module, provides an initial estimation f^0_{approx}, while the deeper levels act as error compensation to gradually improve the overall accuracy. Thus, the run-time configurability can be easily realized by specifying the desired depths. We can then minimize the total error by recursively partitioning the underlying domain into 4 sub-domains and obtain 4^n sub-domains in total for n level approximation, each with an area of $\frac{1}{4^n}$. If we denote $\Delta f_i = f^i_{approx} - f^{i-1}_{approx}$ as the difference between two approximations with 4^i and 4^{i-1} sub-domains, which provides the deviation from a finer granularity partitioning to a coarser one. Then Δf_i is also the output of the i^{th} level error compensation module in Fig. 1. The proposed architecture simply implements the configurability and approximation through the control of partitioning granularity as follows:

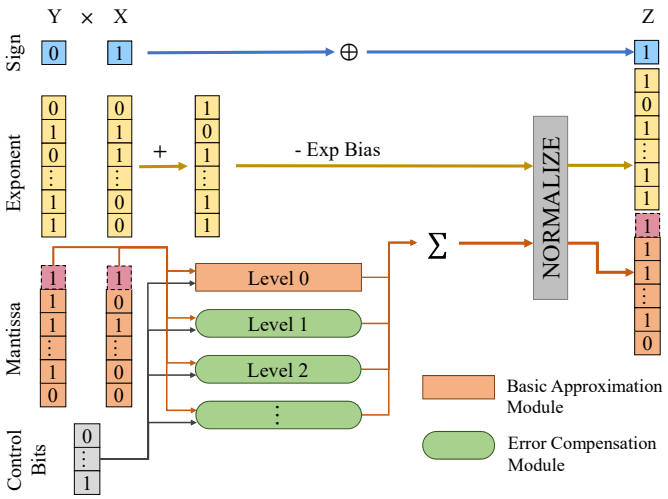

Fig. 1. Architecture of the proposed approximate multiplier.

$$f^n_{approx} = f^0_{approx} + \sum_{i=1}^{n}\Delta f_i. \tag{8}$$

$$\Delta f_n = \begin{cases} 1.5x + 1.5y - 2.25, & n=0,\\ \frac{y[n]?(x):(-x)+x[n]?(y):(-y)}{2^{n+1}} + o_n, & n>0. \end{cases} \tag{9}$$

where the pre-calculated constants o_n

$$o_n = \hat{x_{n-1}} \times \hat{y_{n-1}} - \hat{x_n} \times \hat{y_n}. \tag{10}$$

$\hat{x_n}$is the middle of the interval that x belongs at level n. For example, if $x = 1.01010$ in binary, then $\hat{x_4} = 1.0101\underline{1}$ for the 4^{th} level interval $[1.0101, 1.0110]$. For the constant o_n, since there are 4^n sub-doamins for level n, the circuit implementation cost to store/compute o_n grows exponentially with n, eventually impairing the multiplier efficiency. This is also a common challenge that approximate multipliers using fitted functions have to confront [6]. Thus, we here propose to further optimize the formulation in Eq. (9) to significantly reduce the area complexity.

$$\hat{x_n} - \hat{x_{n-1}} = \frac{x[n]?(1):(-1)}{2^{n+1}}, \tag{11}$$

where "? :" is the conditional operator and $x[n]$ is the n^{th} bit of x. Then Eq.(9) can be simplified as follows:

$$\begin{aligned}\Delta f_n = &\Big\{\Big[\big(x[n]?(1):(-1)\big)\times(y-\hat{y_{n-1}})\Big]\\ &+\Big[\big(y[n]?(1):(-1)\big)\times(x-\hat{x_{n-1}})\Big]\Big\} \gg (n+1)\\ &+\Big[\big(x[n]\oplus y[n]\big)?(1):(-1)\Big] \gg (2n+2).\end{aligned} \tag{12}$$

In Eq. (12), the constant o_n has been merged to 3 additions and 1 XOR operation. Thus, when $n > 0$, the proposed model does not include any explicit multiplication. All the operations

it has are at most 1 XOR operation, 2 arithmetic negations, and 5 additions, which can be implemented with much smaller circuit cost. The complexity for each level can be reduced from the exponential one in Eq. (9) to a constant one.

C. Design for Circuit-Friendly Implementation

The circuit implementation complexity in Eq. (12) can then be further reduced by simplifying the output logic and using a modified tree structure. Since the circuit implementations of FP addition is not trivial in area, it is highly desired to further optimize the addition operations. Eq. (12) can be further simplified a more circuit implementation friendly expression as below (the simplification details are not included here due to the page limit):

$$\begin{aligned} \Delta f_n &= p_{x,n} + p_{y,n} + c_n \\ p_n^x &= \{x[n] \odot y[n:M]\}_{2n+1} + \{\overline{x[n]}\}_{M+n+1} \\ p_n^y &= \{y[n] \odot x[n+1:M]\}_{2n+1} + \{\overline{y[n]}\}_{M+n+1} \\ c_n &= -3/2^{2n+2}, \end{aligned} \tag{13}$$

where $\odot$ denotes the bit-wise exclusive-XOR (*i.e.*, XNOR) between $x[n]$ and each bit of $y[n:M]$; $\overline{x[n]}$ is the logic negation of $x[n]$; $x[i:j]$ is a binary string extracted from the i^{th} to the j^{th} bits of the fractional part of x; and $\{z\}_i$ is a fractional binary obtained by placing z from the i^{th} bit after the decimal point. To be consistent with the conventional multiplier, $p_n^x + p_n^y$ in each level of the proposed multiplier can be considered as **partial product**, which can be implemented as in Fig. 2(a) and then summed up to generate the output. This sub-figure illustrates the dependency between partial products in each level and the inputs. Thus, if we ignore the constant part, **the computation of Eq.** (13) **only involves two addition.**

For the constant part, we can merge all such constants at different levels into one. By simplifying the Level-0 approximation in Eq. (9) to:

$$\Delta f_0 = 1.5(x-1) + 1.5(y-1) + 0.75, \tag{14}$$

we can combine the constant 0.75 together with the other constant terms from the n levels as:

$$0.75 - \sum_{i=1}^{n} \frac{3}{2^{2i+2}} = 0.5 + \frac{1}{2^{2n+2}}. \tag{15}$$

Eq. (15) indicates that, when given the approximation level of n, instead of actually computing Eq. (15), we can directly construct the sum of all the constants by setting the 1^{st} and the $(2n+2)^{th}$ bits to 1, as shown in Fig. 2(b).

With the discussions above, we can now optimize the original OAM architecture in Fig. 1 to a more circuit-friendly implementation, as shown in Fig. 3. Fig. 3 illustrates the dependency of the partial products in each level on the inputs. As shown, There are 5 partial product in the basic approximation level, including 4 which rely on the input and 1 constant offset that only relies the approximate level. Each error compensation level contains 2 partial products that rely both on the input mantissa and the approximate level.

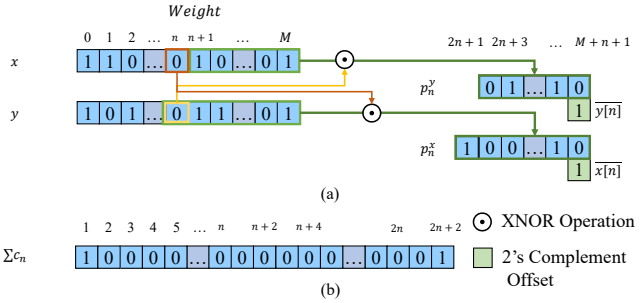

Fig. 2. (a) Partial product generation logic for Level n; (b) Constant computation logic when the approximation level is configured to n.

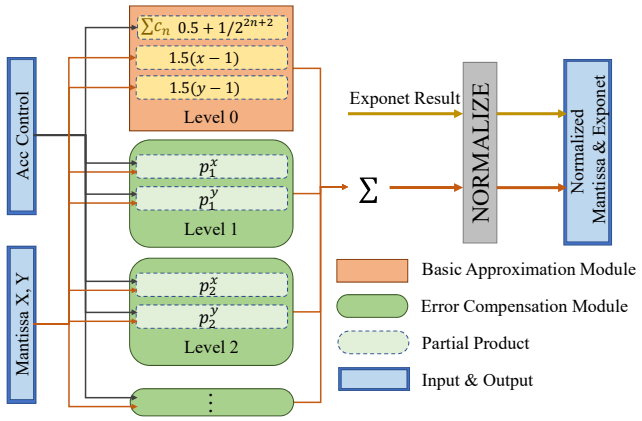

Fig. 3. Optimized architecture for the proposed approximate multiplier.

To speed up the partial products reduction, a tree structure, *e.g.*, Wallace tree, is commonly used. However, a partial product reduction structure designed for an accurate multiplier cannot be directly applied to the proposed approximate multiplier with run-time configurability, as the target approximate level is not fixed. We here propose a new tree structure for partial product reduction in our approximate multiplier, which can switch among different approximate levels within one circuit. An example multiplier configurable among the approximate levels 2, 6, and 11, is shown in Fig. 4, where the dot in the figure represents a partial product, the bracket represents a 3-to-2 compressor, the dotted arrow represents a direct wire transferring the partial product to a latter stage. The tree has three output nodes marked with red rectangles corresponding to the approximation levels of 2, 6, and 11, respectively. As shown in the figure, for a target approximate level lower than 2, the multiplexer selects two summation from the nodes at stage 4. For the approximation level between 2 and 6, the multiplexer takes the other two summations at stage 6. Thus, when working at a lower approximation level, the partial product reduction in Fig. 4 does not need to go through the entire tree structure so as to reduce the critical path delay, which is an appealing feature for run-time configurability. Note that Fig. 4 only shows the structure for the approximation levels of 2, 6, and 11. The proposed structure in Fig. 4 can be easily truncated or modified to adapt to a particular level. When running at lower approximation levels, the tree branches can be power gated to prevent the unnecessary computations for unwanted approximation levels.

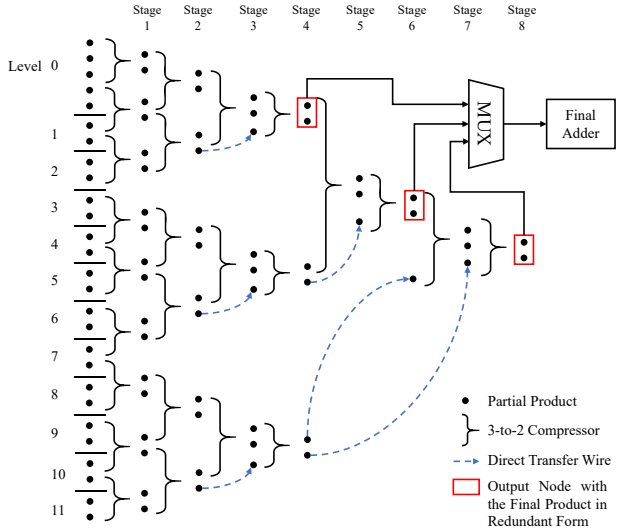

Fig. 4. Partial product reduction tree structure for the proposed multiplier OAM with run-time configurability.

D. Uniqueness

The novelty and uniqueness of this work are summarized as follows:

1) We propose a **theoretically sound optimization formulation** to optimize the approximation error of the approximate multiplier and act as the basis for multiplier architecture design. With the proposed formulation, the error is **symmetrically distributed**, yielding an **unbiased error distribution**.
2) A common issue of the prior approximate FP multiplier is the exponentially growing area complexity with the increased precision requirements. With the proposed architecture, we can reduce the area complexity from $O(4^n)$ to $O(n)$, where n is the number of approximation levels, while ensuring **the same accuracy quality**.
3) We explore detailed optimization for circuit-friendly implementation. With our optimized architecture, the number of partial products has been greatly reduced and all levels can be **simultaneously** invoked and executed. Besides, a modified tree structure is proposed to enable **runtime configurability** and **fast execution**.

IV. RESULTS AND CONTRIBUTIONS

The proposed design is evaluated with both software and hardware implementations. The software implementation can be deployed in various applications like multimedia and neural-networks by replacing the existing FP multiplier unit with the proposed approximate one to measure accuracy. For hardware cost evaluation, we implement the proposed designs in Verilog HDL and then synthesize with *Synopsys* Design Compiler using the UMC 40-nm library. The power and delay are measured using *Synopsys* PrimeTime. In our evaluatios, we first compare the proposed design with ApproxLP [6], which was reported to have SOTA performance in almost all the aspects when compared to the prior work. We then evaluate the software performance in terms of quality and energy efficiency for various applications using the proposed multiplier.

A. Quantitative Comparison to Prior Work

Due to the capability of parallel execution and lower hardware cost, the proposed multiplier has a much smaller delay than ApproxLP with 3.6-24.2% delay improvement for the same approximation level. This is due to the fact that ApprxoLP requires time-consuming comparison before any computation. In addition, the proposed design is also more compact and can eventually achieve 5.5-16.1% area-delay product (ADP) improvement over ApproxLP. Moreover, it is noted that while ApproxLP has an exponentially growing complexity, our design is circuit-implementation-friendly with a linear complexity. Thus, our approximate multiplier can not only support low precision computation like [6], but also be extended to a much higher accuracy level like Levels 6 and 11 to support high-precision computation. When compared to an accurate FP multiplier, we can obtain 33-57% ADP improvement and 56-61% delay improvement. Thus, the proposed multiplier has very good flexibility to support various scenarios with reasonable hardware overhead.

We further compare the accuracy of the proposed design *w.r.t.* ApproxLP for different error measures and different approximation levels. For the same approximation level, the proposed design is able to achieve higher accuracy for all the error measures, with 23–37% accuracy improvement. Fig. 5 shows the error distribution of the proposed design and ApproxLP. The proposed design can achieve a much tighter error distribution with smaller standard deviation than ApproxLP, indicating consistently more accurate approximation. Note that for ApproxLP [6] at Level 0, the distribution is only half-sided and hence results in biasedness in the error distribution.

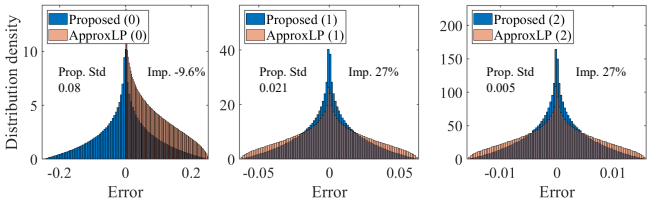

Fig. 5. Error distribution comparison between the proposed multiplier and ApproxLP [6] for different approximation levels (0-2).

B. Application Level Efficiency

We evaluate the efficiency of the proposed multiplier on several multi-media processing and machine learning applications. MNIST(MLP) refers to the execution of a 5-layer multi-layer perceptron (MLP) on MNIST dataset, while MNIST(CNN) and CIFAR-10 refer to the execution of a pre-trained AlexNet on MNIST and CIFAR-10 datasets, respectively. The embedded approximate multiplier is configured to 4 levels of approximation (Level 0, 1, 2, 6). The reported energy consumption of the multipliers are obtained from PrimeTime PX. Multi-media processing tasks are evaluated with the metric of Peak Signal to Noise Ratio (PSNR), while machine learning tasks use accuracy loss as the accuracy measure.

Table I reports PSNRs and accuracy losses for all the tasks. Machine learning tasks have a strong resilience to the

TABLE I
COMPARISON OF PSNR/ACCURACY LOSS, ENERGY AND EDP IMPROVEMENTS FOR DIFFERENT OPENCL, MULTI-MEDIA PROCESSING AND MACHINE LEARNING TASKS USING THE PROPOSED APPROXIMATE MULTIPLIER WITH DIFFERENT LEVELS OF APPROXIMATIONS.

Application	PSNR(dB)/Accuracy Loss(%)				Energy Improvement				EDP Improvement			
	Level 0	Level 1	Level 2	Level 6	Level 0	Level 1	Level 2	Level 6	Level 0	Level 1	Level 2	Level 6
Random Input	45.66	57.71	69.74	117.9	7.58 ×	5.03 ×	3.43 ×	1.77 ×	21.77 ×	11.98 ×	7.89 ×	3.19 ×
Audio FFT	63.32	76.60	89.54	138.1	7.79 ×	4.20 ×	2.83 ×	1.37 ×	22.37 ×	10.00 ×	6.49 ×	2.47 ×
RGB-to-Gray	35.52	53.42	61.32	105.6	3.05 ×	2.03 ×	1.24 ×	0.64 ×	8.76 ×	4.84 ×	2.84 ×	1.15 ×
Gauss Blur	49.69	56.68	64.55	111.1	2.40 ×	1.42 ×	0.87 ×	0.43 ×	6.90 ×	3.38 ×	2.01 ×	0.78 ×
CIFAR-10	0.29%	0.01%	0%	0%	7.12 ×	4.62 ×	3.19 ×	1.63 ×	20.46 ×	11.00 ×	7.33 ×	2.93 ×
MNIST(CNN)	0.02%	0%	0%	0%	6.72 ×	4.52 ×	3.25 ×	1.67 ×	19.31 ×	10.76 ×	7.47 ×	3.01 ×
MNIST(MLP)	0.11%	0.02%	0%	0%	6.80 ×	4.58 ×	3.28 ×	1.67 ×	19.54 ×	10.89 ×	7.53 ×	3.00 ×

errors with consistently small accuracy loss. From the table, even with Level 1 approximation, we can achieve 39-57dB for the multi-media processing tasks and almost negligible accuracy loss (0-0.02%) for machine learning tasks. A higher approximation level, *e.g.*, Level 2, can further improve the PSNR and accuracy loss to almost 0%.

Table I also compares the normalized energy and energy-delay-product (EDP) improvements by the proposed multiplier in comparison to a full-precision FP multiplier. For the tasks like Gauss Blur, one input to the multiplier is almost kept constant during the execution, which simply implies there are fewer non-zero bits, and results in fewer bit switching and energy consumption. Thus, for those tasks, EDP improvements by the proposed multiplier are limited, with only 6.9×, 3.38×, and 2.01× on average for Level 0, 1 and 2, respectively. For the tasks like machine learning and audio processing, they have more varying inputs and non-zeros in the FP numbers. This significantly increases the bit switching activity and hence the power consumption, eventually resulting in huge benefits using the proposed multiplier. In particular, for CIFAR-10, the average EDP improvements for Level 0, 1 and 2 approximation can reach 20.46×, 11×, and 7.33×, respectively.

C. Contributions

After 2-year research in this area, the efforts have resulted in **2 first-author papers published in the premier EDA conferences (ICCAD'20 and SoCC'20), 1 granted patent** and 1 transaction paper in preparation. This work was not only awarded the Golden Metal in ACM/ICCAD SRC but also **received the ICCAD'20 Best Paper Award nomination (6 out of 470 papers)**. To conclude, this work proposed an efficient run-time configurable approximate multiplier. The multi-level architecture can easily incorporate the run-time configurability without incurring much area overhead, and naturally reach optimal approximation with unbiased error distribution. Our evaluations on various aspects show that the proposed design has comprehensive advantages over prior multiplier designs and is able to outperform SOTA design in terms of accuracy, area, and delay.

V. PUBLICATIONS AND PATENTS

[1] **C. Chen**, S. Yang, W. Qian, M. Imani, X. Yin, and C. Zhuo. Optimally Approximated and Unbiased Floating-Point Multiplier with Runtime Configurability. In IEEE/ACM ICCAD 2020, **Nominated for Best Paper Award**.

[2] **C. Chen**, Q. Huang, C. Li, L. Zhang, C. Zhuo, and X. Yin. Analog Content Addressable Memory Using Ferroelectric: A Case Study of Search-In-Memory. In IEEE SoCC 2020.

[3] C. Zhuo, **C. Chen** and S. Yang. An Unbiased Approximate Multiplier for Normalized Floating-Point Numbers and Its Implementation Method. Granted patent: ZL202010969041.7. 2020.

[4] **C. Chen**, W. Qian, M. Imani, X. Yin, and C. Zhuo. OAM: An Optimally Approximated Floating-Point Multiplier with Unbiasedness and Configurability. In Preparation for IEEE TC.

REFERENCES

[1] P. J. Song and G. De Micheli, "Circuit and architecture trade-offs for high-speed multiplication," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 9, pp. 1184–1198, 1991.

[2] R. K. Yu and G. B. Zyner, "167 mhz radix-4 floating point multiplier," in *Proceedings of the 12th Symposium on Computer Arithmetic*, 1995, pp. 149–154.

[3] S. Palekar and N. Narkhede, "High speed and area efficient single precision floating point arithmetic unit," in *2016 IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, 2016, pp. 1950–1954.

[4] 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*, ser. ISLPED '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3218603.3218621

[5] 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)*, 2017, pp. 1–6.

[6] 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 *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[7] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *2011 24th Internatioal Conference on VLSI Design*, 2011, pp. 346–351.

[8] V. Camus, J. Schlachter, C. Enz, M. Gautschi, and F. K. Gurkaynak, "Approximate 32-bit floating-point unit design with 53% power-area product reduction," in *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference*, 2016, pp. 465–468.

[9] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim, "Energy-efficient approximate multiplication for digital signal processing and classification applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 6, pp. 1180–1184, 2015.

[10] C. Liu, J. Han, and F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2014, pp. 1–4.

[11] K. Bhardwaj, P. S. Mane, and J. Henkel, "Power- and area-efficient approximate wallace tree multiplier for error-resilient systems," in *Fifteenth International Symposium on Quality Electronic Design*, 2014, pp. 263–269.

[12] C. Lin and I. Lin, "High accuracy approximate multiplier with error correction," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, 2013, pp. 33–38.

[13] S. Hashemi, R. I. Bahar, and S. Reda, "Drum: A dynamic range unbiased multiplier for approximate applications," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 418–425.