Optimally Approximated and Unbiased Floating-Point Multiplier with Runtime Configurability

Chuangtao Chen[2], Sen Yang[1], Weikang Qian[4], Mohsen Imani[5], Xunzhao Yin[1], Cheng Zhuo[1,3]*
[1]College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China
[2]College of Electrical Engineering, Zhejiang University
[3]ASIC & System Key Lab, School of Microelectronics, Fudan University, Shanghai, China
[4]University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University
[5] Department of Computer Science and Engineering, University of California, Irvine
*E-mail: czhuo@zju.edu.cn

ABSTRACT

Approximate computing is a promising alternative to improve energy efficiency for IoT devices on the edge. This work proposes an optimally approximated and unbiased floating-point approximate multiplier with runtime configurability. We provide a theoretically sound formulation that turns multiplication approximation to an optimization problem. With the formulation and findings, a multi-level architecture is proposed to easily incorporate runtime configurability and module execution parallelism. Finally, an optimization scheme is applied to improve the area, making it linearly dependent on the precision, instead of quadratically or exponentially as in prior work. In addition to the optimal approximation and configurability, the proposed design has an efficient circuit implementation that uses inversion, shift and addition instead of complex arithmetic operations. When compared to the prior state-of-the-art approximate floating-point multiplier, *ApproxLP* [30], the proposed design outperforms in all aspects including accuracy, area, and delay. By replacing the regular full-precision multiplier in GPU, the proposed design can improve the energy efficiency for various edge computing tasks. Even with Level 1 approximation, the proposed design improves energy efficiency up to 122× for machine learning on CIFAR-10, with almost negligible accuracy loss.

1 Introduction

Due to the rapid growth of Internet-of-Things (IoT), energy efficiency has become a critical concern, especially when IoT devices are deployed with constrained resources [1-4]. There have been various research efforts to optimize energy efficiency for IoT devices from algorithm, architecture, to circuit [5-15]. Among such efforts, approximate computing has emerged as a promising alternative for designers to trade computational accuracy with energy efficiency. This is especially applicable to human sensory or machine learning tasks where a small amount of inaccuracy is tolerable or even ignorable [16-19].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
ICCAD '20, November 2–5, 2020, Virtual Event, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8026-3/20/11…$15.00
https://doi.org/10.1145/3400302.3415702

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 pre-determined but sometimes unnecessary precisions, inevitably degrading their energy efficiency. When running data-intensive applications, e.g., image processing or machine learning, due to the large range of input operands, most conventional processors heavily rely on floating-point units (FPU) [7, 21]. To cover the same dynamic range, the fixed-point unit demands up to 5x larger area compared to its FP counterpart and hence is a far less common option [22]. Among different FP operations, multiplication is widely used but possibly the most energy consuming operation for various data-intensive scenarios, such as streaming, neural network, image processing, etc. In other words, when running inaccuracy-tolerable applications on the conventional processors, significant energy and time are spent on FP multipliers computing highly accurate outputs that are not necessarily demanded. Thus, for FP multiplication in IoT devices, there is a need to optimize its energy efficiency by *providing sufficient instead of excessively accurate computational precisions*.

As a common arithmetic component that has been studied for decades [23, 24], the past focus for FP multiplier is mainly placed upon accuracy and performance. Recently, with awareness of the compromise between the stringent resource constraint and the accuracy tolerance for edge applications, researchers have growing interests in designing an approximate FP multiplier to improve energy efficiency. For example, Camus *et al.* redesigned major arithmetic components to reduce circuitry complexity [25], where the approximation error is controlled by construction. Works in [26, 27] use a hybrid method by employing both accurate and inaccurate multipliers for runtime configurable approximation. However, such FP multipliers can *hardly guarantee unbiased error distribution with near-zero average error*, causing the risk of aggregated error for applications with multiple multiplications in series.

To address the issues, several works propose to design approximate multipliers at the algorithmic level to achieve configurability by combining different product sizes or truncating unwanted bits [28, 29]. Recently, some work proposed to improve computational efficiency and configurability by directly approximating the product of two FP inputs with linear fitting [30]. However, due to the focus at algorithmic level, the proposed approaches may suffer from quickly increased circuitry complexity and degraded efficiency with higher precision requirements, eventually impairing energy

efficiency and computation time. More importantly, for the required precision and configuration, is the proposed approximation the best we can have? Many designs happen to rely on hand-crafted structures or heuristics. How to achieve an optimal approximation with unbiased error distribution remains an open question. Thus, it is *highly desired to develop a systematic methodology to design unbiased, configurable, and circuit-implementation-friendly FP multiplier with optimal approximation.*

Apparently, this is not a trivial task: (1) On one hand, unlike the many approximations in prior work that stem from heuristic findings [12, 26, 28, 30], we need to formally define the problem, including objective function and constraints, to *enable the theoretically sound basis for optimal approximation.* (2) On the other hand, when ensuring configurability, the underlying architecture should *facilitate the circuitry implementation* instead of introducing implementation-unfriendly logics or operations, to prevent exponentially growing area complexity with higher precision requirements. (3) Finally, how to *ensure unbiasedness and tunability* for the optimally approximated FP multiplier is not straightforward. It is hard to achieve all the features in one design.

Thus, in this paper, by addressing the aforementioned challenges we propose to design a *runtime configurable FP multiplier* that is *optimally approximated with unbiased error distribution*. The major contributions of our work are listed as follows.

- **A theoretically sound optimization formulation is proposed** to optimize the approximation error of the approximate multiplier and act as the basis for multiplier architecture design. With the proposed formulation, the error can be symmetrically distributed, yielding an unbiased error distribution.
- Based on the optimization formulation and findings, we propose **a multi-level FP multiplier architecture that can easily incorporate run-time configurability**. The accuracy is configured by adding up different levels of error compensations, while each level of compensation is designed with circuit-implementation-friendly operations, such as shifting, inversion, and addition. Moreover, **the modules at different levels are independent and hence support parallel execution** to achieve higher efficiency.
- A common issue of the prior approximate FP multiplier designs is the quickly growing area complexity with the increased precision requirements. With the proposed architecture, we theoretically analyze the cost complexity and **propose an optimization scheme to reduce the complexity from $O(4^n)$ to $O(n)$**, where n is the number of approximation levels, while ensuring the same accuracy quality.

Experimental results show that, with the proposed formulation to determine the optimal approximation, we can implement an energy-efficient and configurable approximate multiplier. The proposed multiplier is found to have comprehensive superiority over many prior work [12, 26-28, 30]. When compared with a state-of-the-art (SOTA) multiplier, the proposal can achieve accuracy improvements up to 37% in terms of mean square error (MSE) with far smaller area (84% saving) and delay (43% improvement). In addition, when replacing a regular FP multiplier with the proposed multiplier and evaluating with various edge-application tasks, we ca achieve 1.8-83.3× energy improvement and 2.4-132.1× energy efficiency improvement while the quality or accuracy loss is almost negligible.

2 Background

2.1 Floating-Point Multiplication

Compared to integer computing, FP arithmetic is usually costlier and energy consuming, due to its complexity. IEEE 754 standard is a technical standard for FP arithmetic [31]. According to it, an FP number consists of sign, exponent and mantissa, as shown in Fig. 1(a). The mantissa of a normalized FP number is defined as integer 1 plus the fractional portion, whose exact value is between 1 and 2. In a general-purpose processor, for an FP multiplication, as shown in Fig. 1(b), the sign bits is computed by an XOR operation, and the exponent bits are computed by an adder. Then the bias is subtracted from the exponent to allow both negative and positive values. Finally, the product is shifted to the range of 1 and 2 to obtain the final result.

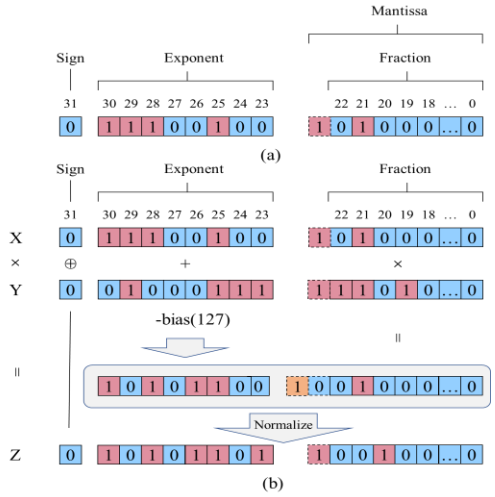

Figure 1: (a) Representation of a 32-bit FP number according to IEEE 754; (b) FP multiplication in a general-purpose processor.

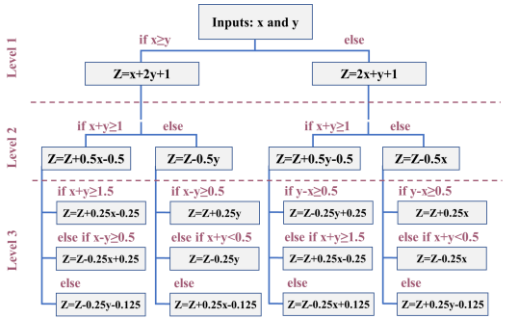

Figure 2: Flow of the approximate multiplier

2.2 Approximate Multiplier

Approximate arithmetic has been a popular research area in the past decade. While multiplier itself is complicated, most prior work 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 use approximate components, such as adders, to build the multiplier, so as to speed up addition or partial product generation [12, 25, 29, 32-34]. To approximate from a higher design level, [35] proposed a pipelined log-based approximation using the classical Mitchell multiplier with an iterative procedure to improve accuracy. To speed up the iterative procedure, researchers propose to truncate the bits after the leading one to conserve energy or utilize a hybrid method with both inaccurate and accurate multipliers to adjust the computational accuracy by selecting the appropriate multiplier, thereby trading off between accuracy and cost [26-28].

However, there are several issues of directly applying the prior work to the IoT devices at the edge. While those methods can precisely control the error, it is hard for many of them to *guarantee unbiased output with zero-mean error distribution*. On the other hand, *configurability is highly demanded for versatile edge scenarios*. The limitation of many prior approaches is either lack of configurability, or the notably high cost to implement such configurability with higher precision requirements.

Recently, *ApproxLP* is proposed to approximate the mantissa product using linear fitting [30]. The design shows much higher performance for the given error rate when compared to the prior approximate multiplier solutions, which is hence considered as a state-of-the-art (SOTA) FP multiplier with significant advantages over prior approximation methods. Fig. 2 describes the basic concept of *ApproxLP*. As shown in the flow, the ranges of the two mantissas are first partitioned into multiple sub-regions, with linear functions introduced to fit each sub-region. The partitioning can be further fine-grained to deeper levels to improve the overall accuracy at the cost of area and delay. The sum of the outputs at each level gradually approaches the exact multiplication product, so that the accuracy can be runtime-configured by enabling different levels. However, while *ApproxLP* improves the efficiency compared with the prior approximation works, it still does not fully address the aforementioned challenges of large implementation cost and biased output error. For example, the error distribution of level 1 approximation in *ApproxLP is biased which may cause error accumulation* with multiple multiplications in series. The *branching for sub-region selection is also hardware-demanding*, causing significantly more area with deeper levels. In addition, as the proposed fitted functions are heuristically customized, it raises a very natural question whether we can achieve more optimal approximation through more theoretically sound formulation. Thus, it is highly motivated for us to fully overcome the existing issues in the prior work and provide the capability to design optimally approximated and unbiased FP multiplier with low hardware cost and runtime configurability.

3 Design and Optimization of Approximate FP Multiplier

With the aforementioned goals, we would like to tackle the challenges with the following steps: (1) Formally formulate the problem of approximated multiplication that can incorporate desired design targets; (2) Propose a multiplier architecture that can facilitate runtime configurability with low hardware cost; (3) Optimize the circuit to improve the overall efficiency.

3.1 Problem Formulation

As shown in Fig. 1, the key operation of an FP multiplication is the product of the two mantissas. We define an FP multiplication as: $z = xy$, where z is the output, and x and y are the input mantissas within the range of $[1, 2)$. A common solution to approximate a function is to project it to another space with equal or lower dimension for simplification. Without loss of generality, we can define the bases of the space as $\{1, x, y, x^2, y^2\}$ and the following inner product for the space to measure the distance of two functions:

$$\langle \boldsymbol{f}, \boldsymbol{g} \rangle = \int_{x_1}^{x_2} \int_{y_1}^{y_2} \boldsymbol{f} \times \boldsymbol{g}\, \boldsymbol{dxdy} \quad (\mathbf{1})$$

where x_1, x_2, y_1, y_2 are the constants that define the input domain of f and g. When $x_2 > x_1 \geq 0, y_2 > y_1 \geq 0$, we can easily prove that the bases $1, x, y, x^2, y^2$ are linearly independent.

To project the multiplication z to the above inner space, we define the following approximate function: $z_{approx} = k_0 + k_1 x + k_2 y + k_3 x^2 + k_4 y^2$, which is a linear combination of the bases. We further define an error measure within the domain $[x_1, x_2] \times [y_1, y_2]$. A mathematically friendly choice is the square error defined below:

$$\left\| z - z_{approx} \right\|^2 = \langle z - z_{approx}, z - z_{approx} \rangle \quad (2)$$

By minimizing the square error, we minimize the deviation between the original and the projected functions.

When there are no additional constraints, this unconstrained problem can be easily solved to obtain the following solution:

$$[k_0, k_1, k_2, k_3, k_4] = [-\tfrac{(x_1+x_2)(y_1+y_2)}{4}, \tfrac{y_1+y_2}{2}, \tfrac{x_1+x_2}{2}, 0, 0] \quad (3)$$

Note that the formulation above is not limited to square error measure, but applicable to different targets or measures for optimization.

Now for $z = xy$ defined on $[x_1, x_2] \times [y_1, y_2]$, we can optimally approximate it by the following linear function according to Eq. (3):

$$z = xy \approx z_{approx} = k_0 + k_1 x + k_2 y \quad (4)$$

The following lemmas state some properties for the approximation by Eq. (4).

***Lemma* 1.** For $z = xy$ in the domain of $[x_1, x_2] \times [y_1, y_2]$, the maximum absolute error by the approximation shown in Eq. (4) is reached when $\{x, y\}$=$\{x_1, y_1\}$, $\{x_1, y_2\}$, $\{x_2, y_1\}$, or $\{x_2, y_2\}$.

Proof:

The partial derivatives of $(z - z_{approx})$ *w.r.t.* x or y are:

$$\frac{\partial(z - z_{approx})}{\partial x} = y - k_1 \quad \text{or} \quad \frac{\partial(z - z_{approx})}{\partial y} = x - k_2 \quad (5)$$

We divide the domain $[x_1, x_2] \times [y_1, y_2]$ into four sub-regions: $R_1 = [x_1, k_2] \times [y_1, k_1]$, $R_2 = [x_1, k_2] \times [k_1, y_2]$, $R_3 = [k_2, x_2] \times [y_1, k_1]$, and $R_3 = [k_2, x_2] \times [k_1, y_2]$. In each sub-region, the derivatives are constantly ≥ 0 or ≤ 0,, which simply means the maximum error of each sub-region always lies at its corners. Then we can find the maximum error in the domain $[x_1, x_2] \times [y_1, y_2]$ by comparing the errors at all 9 corners. Since the five corners in the center happen to be 0, the maximum absolute errors are then

reached at the four corners of $[x_1, x_2] \times [y_1, y_2]$, i.e., $\{x_1, y_1\}$, $\{x_1, y_2\}$, $\{x_2, y_1\}$, or $\{x_2, y_2\}$.

***Lemma* 2**. For $z = xy$ in the domain of $[x_1, x_2] \times [y_1, y_2]$, the approximation by Eq. (4) is unbiased, *i.e.*, the mean of error distribution is 0 for uniformly distributed inputs.

Proof:

When the inputs are uniformly distributed in the domain of $[x_1, x_2] \times [y_1, y_2]$, by Eq. (3), the mean of error distribution is:

$$\iint z - z_{approx} dxdy = \int_{x_1}^{x_2} \int_{y_1}^{y_2} xy - (k_0 + k_1 x + k_2 y) dxdy = 0 \quad (6)$$

This implies that the approximation by Eq. (4) is unbiased

***Lemma* 3**. For a given number of sub-regions partitioned from the domain of $[x_1, x_2] \times [y_1, y_2]$, with one approximate function in each sub-region, the total square error of approximation is minimized when each sub-region contains exactly the same area.

Proof:

The square error for the domain $[x_1, x_2] \times [y_1, y_2]$ is:

$$\|z - z_{approx}\|^2 = \frac{(x_1 - x_2)^3 (y_1 - y_2)^3}{144} = \frac{S^3}{144} \quad (7)$$

where $S = (x_1 - x_2)(y_1 - y_2)$ is the area of the region. Suppose that we partition $[x_1, x_2] \times [y_1, y_2]$ into n sub-regions, with the total area of all the sub-regions equal to S, *i.e.* $\Sigma_i s_i = S$, where s_i is the area of the i^{th} sub-region.

Within each sub-region, we can compute a fitted approximation using Eqs. (3) and (4), reaching a square error of ${s_i}^3/144$. The total square error for all the sub-regions are simply $\Sigma {s_i}^3/144$. According to the generalized mean inequality, we have:

$$\sqrt[3]{\frac{\Sigma {s_i}^3}{n}} \geq \frac{\Sigma s_i}{n} = \frac{S}{n} \quad (9)$$

Equality is reached if and only if:

$$s_1 = s_2 = \cdots = s_i = \cdots = s_n = \frac{S}{n} \quad (10)$$

yielding the following minimum square error for n sub-regions:

$$\text{Min}\left(\frac{\Sigma {s_i}^3}{144}\right) = \frac{S^3}{144 n^2} \quad (11)$$

Eq. (11) also implies that, *with more fine-grained partitioning into smaller sub-regions and computing approximations within each sub-region, the total square error can be further reduced*.

3.3 Architecture for Multi-Level Approximate Multiplier with Runtime Configurability

From the derivations in the last subsection, we can observe that: (1) According to *Lemma 2*, the approximation by Eq. (4) on a rectangle domain naturally has unbiased error distribution; (2) According to *Lemma 3*, the finer granularity of partition yields to smaller approximation error, which can support the design of a configurable multiplier. Thus, we here propose *a multi-level approximate multiplier architecture* that is runtime configurable. Fig. 3 shows the proposed architecture with a multi-level structure. As shown in the figure, **Level 0 is denoted as the basic approximation module, which provides an initial estimation z^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.

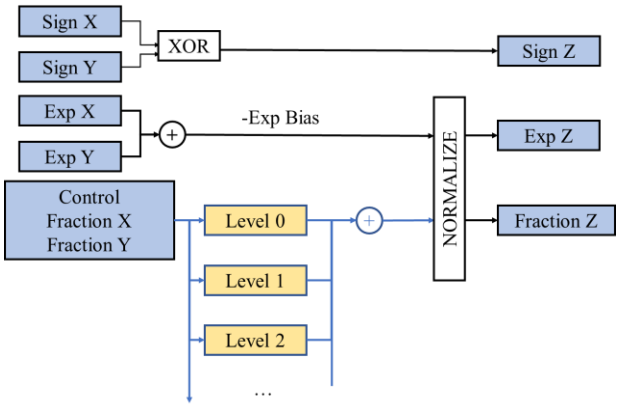

Figure 3: Architecture of the proposed approximate multiplier.

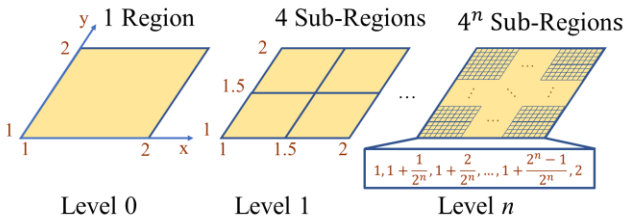

Figure 4: Procedure of partitioning for the proposed multiplier.

As discussed in Sec. 2, the initial domain for the mantissas in an FP multiplication is $[1,2) \times [1,2)$. According to the lemmas in the last subsection, for one level deeper, we can partition the underlying domain to 2^2 sub-regions of rectangle shapes. Thus, if the multiplier is configured with a depth of n, as shown in Fig. 4, we can recursively partition the underlying domain and obtain 4^n sub-regions, each with an area of $\frac{1}{4^n}$. In other words, **the approximation after the i_{th} partition, denoted as z^i_{approx} for Level i, is a piecewise linear model that contains 4^i linear models as shown in Eq. (4)**, one corresponding to a rectangular sub-region.

Thus, the error compensation simply provides the deviation from a finer granularity partitioning to a coarser one. If we denote $\Delta z_i = z^i_{approx} - z^{i-1}_{approx}$ as the difference between two approximations with 4^i and 4^{i-1} sub-regions, then Δz_i is also the output of the i_{th} level error compensation module. The proposed architecture in Fig. 3 simply implements the configurability and approximation through the control of partitioning granularity as follows:

$$z^n_{approx} = \sum_{i=1}^{n} \Delta z_i + z^0_{approx} \quad (12)$$

For the initial estimation of z^0_{approx} on $[1,2) \times [1,2)$, it can be easily computed by Eqs. (3) and (4) as:

$$z^0_{approx} = 1.5x + 1.5y - 2.25 \quad (13)$$

where x and y are the two input mantissas.

The remaining question of the proposed architecture is how to realize the error compensation Δz_i for each level. Here we propose a simple but circuit-implementation-friendly model to efficiently

compute the error compensation. Note that in Eq. (3), the found coefficients of k_1 and k_2 are just the coordinates of the interval center, and k_0 is the negated product of k_1 and k_2. Such properties can be leveraged to enable easy implementation of error compensation. As in Fig. 4, if we denote the k_{th} interval for x on the i_{th} level as $[x^i_{k-1}, x^i_k]$, the interval center (middle point) is simply:

$$\widehat{x^i_k} = x^i_{k-1} + \frac{1}{2^{i+1}} \tag{14}$$

Note that if representing x^i_{k-1} in a binary format, its decimal part just needs i bits. Thus, $\widehat{x^i_k}$ simply attaches an additional bit '1' at the end of x^i_{k-1}. Fig. 5 gives a simple example for the 6th interval on the 4th level to illustrate the above discussion, where x=1.01010…0 is an input mantissa, x^4_5=1.0101 is the left bound of the 6th interval, $\widehat{x^4_5}$=1.01011 is the interval center. For y-axis, we can reach a similar construction method.

Figure 5: An example for the 6th interval on the 4th level.

Then, by Eq. (3), we can derive the following for any level n:

$$z^n_{approx} = \widehat{x^n_{p(n)}} \times y + \widehat{y^n_{q(n)}} \times x - \widehat{x^n_{p(n)}} \times \widehat{y^n_{q(n)}} \tag{15}$$

$$\Delta z_n = \left(\widehat{x^n_{p(n)}} - \widehat{x^{n-1}_{p(n-1)}}\right) \times y + \left(\widehat{y^n_{q(n)}} - \widehat{y^{n-1}_{q(n-1)}}\right) \times x + o^n_{p,q} \tag{16}$$

where $p(n), q(n)$ are the indexes to specify the subregions of x and y in the n_{th} level; $o^n_{p,q}$ is a constant that can be pre-calculated as below for each sub-region:

$$o^n_{p,q} = \widehat{x^{n-1}_{p(n-1)}} \times \widehat{y^{n-1}_{q(n-1)}} - \widehat{x^n_{p(n)}} \times \widehat{y^n_{q(n)}} \tag{17}$$

The formulations above still seem complex but can be further simplified to more circuit-implementation-friendly format by noting the characteristics when representing in binaries. As in Fig. 5, if we denote n_{th} bit of the input mantissa x as $x[n]$, we have the following:

$$\widehat{x^n_{p(n)}} - \widehat{x^{n-1}_{p(n-1)}} = \frac{x[n]?(1):(-1)}{2^{n+1}} \tag{18}$$

The absolute difference between $\widehat{x^n_{p(n)}}$ and $\widehat{x^{n-1}_{p(n-1)}}$ is always $1/2^{n+1}$, while the sign is determined by $x[n]$. In other words, **Eq. (18) can be easily implemented in circuit with 1 $(n+1)$-bit right shift operation and at most 1 inversion operation.**

In summary, with the pre-calculated constants $o^n_{p,q}$ (that can be stored in RAM or hard-coded in circuit) by Eq. (17), we can achieve the following circuit-implementation-friendly models for the proposed architecture:

$$\Delta z_n = \begin{cases} 1.5x + 1.5y - 2.25 & n = 0 \\ \frac{\{y[n]?(1):(-1)\}x + \{x[n]?(1):(-1)\}y}{2^{n+1}} + o^n_{p,q} & n > 0 \end{cases} \tag{19}$$

Apparently in eq. (19), when n>0, the model does not include any multiplication. **All the operations it has are: 2 inversion (at most), 1 n+1-bit shift, and 2 additions,** which can be implemented with much smaller circuit cost.

3.4 Optimization for Complexity Reduction

As described in the last sub-section, each sub-region has a constant $o^n_{p,q}$. Since there are 4^n sub-regions for level n, the circuit implementation cost to store/compute $o^n_{p,q}$ *grows exponentially with n, eventually impairing the multiplier efficiency*. This is also a common challenge that most configurable multipliers have to confront [26, 27, 30]. Thus, we further optimize the formulation in Eq. (19) to significantly reduce the complexity in area.

With Eq. (18), we can rewrite Eq. (17) to the following:

$$\begin{aligned} o^n_{p,q} &= \widehat{x^{n-1}_{p(n-1)}} \times \widehat{y^{n-1}_{q(n-1)}} - \widehat{x^n_{p(n)}} \times \widehat{y^n_{q(n)}} \\ &= \widehat{x^{n-1}_{p(n-1)}} \times \widehat{y^{n-1}_{q(n-1)}} - \left(\widehat{x^{n-1}_{p(n-1)}} + \frac{x[n]?(1):(-1)}{2^{n+1}}\right) \\ &\quad \times \left(\widehat{y^{n-1}_{q(n-1)}} + \frac{y[n]?(1):(-1)}{2^{n+1}}\right) \end{aligned} \tag{20}$$

This equation can be further simplified to the following, where $\gg$ denotes the shift operation:

$$\begin{aligned} o^n_{p,q} = &-[(x[n]?(1):(-1)) \times \widehat{y^{n-1}_{q(n-1)}}] \gg (n+1) \\ &-[(y[n]?(1):(-1)) \times \widehat{x^{n-1}_{p(n-1)}}] \gg (n+1) \\ &+[(x[n]\hat{}y[n])?(1):(-1)] \gg (2n+2) \end{aligned} \tag{21}$$

where ^ denotes XOR operation.

Then, Eq. (19) for n>0 can be refined to the following:

$$\begin{aligned} \Delta z_n = \{&\left[(x[n]?(1):(-1)) \times \left(y - \widehat{y^{n-1}_{q(n-1)}}\right)\right] \\ &+ \left[(y[n]?(1):(-1)) \times \left(x - \widehat{x^{n-1}_{p(n-1)}}\right)\right]\} \gg (n+1) \\ &+[(x[n]\hat{}y[n])?(1):(-1)] \gg (2n+2) \end{aligned} \tag{22}$$

Note that $\widehat{x^{n-1}_{p(n-1)}}$ is the middle point of the interval that contains x. The subtraction $x - \widehat{x^{n-1}_{p(n-1)}}$ can be implemented with logic operations by turning the first n bits in x to 0 and bit flipping (if necessary) for the rest of bits instead of arithmetic operations. Thus, with the improved formulation shown in Eq. (22), the approximation computation for an input pair involves 1 XOR, 1 shift, and 2 additions as well as several inversion and logic operations. **The area cost is hence linear with the depth n for higher precisions instead of exponential or quadratic dependence as in prior work**.

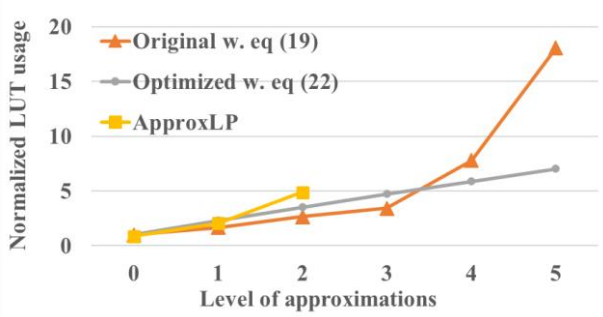

Figure 6: Complexity comparison in LUT usage for *ApproxLP* [30], the proposed multiplier implemented using Eq. (19), and the optimized multiplier implemented using Eq. (22).

Fig. 6 compares the normalized LUT usages among *ApproxLP* [30], the proposed multiplier implemented using Eq. (19), and the optimized multiplier implemented using Eq. (22). Note that in [30],

ApproxLP only provides the fitted model coefficients for the first three levels, which already shows nonlinear trend. The comparison validates the linear complexity of the optimized multiplier *w.r.t.* the approximation levels, *i.e.* depth *n*. Moreover, it is noted that for smaller level of approximations, the implementation by Eq. (19) incurs smaller area overhead by hard-coding the constants. Since the modules of the proposed architecture at different levels does not depend on each other, we can employ a hybrid implementation using both Eqs. (19) and (22) to further minimize the area.

3.5 Additional Features

In addition to the discussion above, there are a few additional features that we can use to facilitate the implementation efficiency:

- The computation for each level in the proposed multiplier is completely independent. Thus, when the number of levels *n* is determined, the modules at different levels can be simultaneously invoked and executed.
- For each module, we can implement it by Eq. (19) or Eq. (22). Most operations involved have a very efficient circuit implementation, such as inversion, shift and addition.
- Unlike ApproxLP that needs complex arithmetic to determine the sub-region, *i.e.*, $x + y \geq 1$, and has dependence on the lower level modules, the proposal can separately evaluate the sub-regions that *x* and *y* belong to, and execute modules at different levels in parallel.
- To determine which sub-region the input pair is located, the proposed module at level *n* only requires the first *n* bits instead of the complete number.

4 Error Analysis

In this section, we further discuss the error distribution for different levels of approximations in the proposed multiplier. Again, here we assume the mantissas of the two inputs *x* and *y* are uniformly distributed within [1, 2). We check all the possible pairs of *x* and *y* in FP representations, and normalize the deviation *w.r.t.* the accurate product, in order to fully investigate if the results are consistent with our theoretical derivations.

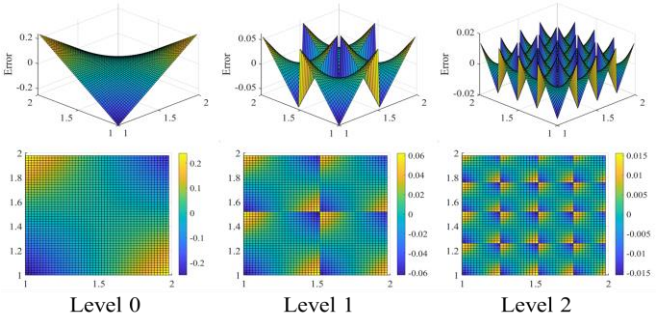

Figure 7: 2-D error distribution for different approximation levels.

Fig. 7 shows the axonometric and top views of the error distributions for Level 0, 1, and 2 approximations. From Fig. 7, we can see that, for each sub-region, the distribution is saddle shaped, with the extreme values reached at the corners of each sub-region. In addition, the unbiasedness is achieved due to the symmetricity of the error distribution. The maximum error quickly drops with a factor close to 4 when using more levels of error compensation. We then measure the performance of the proposed design for different error measures, as shown in Fig. 8, which plots 1-D view for clarity. Two error measures, mean square error (MSE) and maximum absolute error (MAE) are used. Like Fig. 7, we can see that both measures yield to similar error distribution but with different error reduction rate *w.r.t.* the approximation level. In Fig. 9, we further compare the envelope of the error distribution histograms for different levels of approximation. It is clear that when configured to a deeper level in the multiplier, *i.e.*, with more error compensation, most calculations by the proposed design have almost zero errors. With the derivations in the prior sections, we can easily compute the relationship between the error measures and the number of levels used in the proposed multiplier. For an arbitrary depth *n*, we can achieve the following for MSE and MAE:

$$MSE = \frac{1}{9\times 16^{n+1}}, \quad MAE = \frac{1}{4^{n+2}} \tag{23}$$

This can be used *as the error control mechanism to configure the desired precision of the multiplier at runtime.*

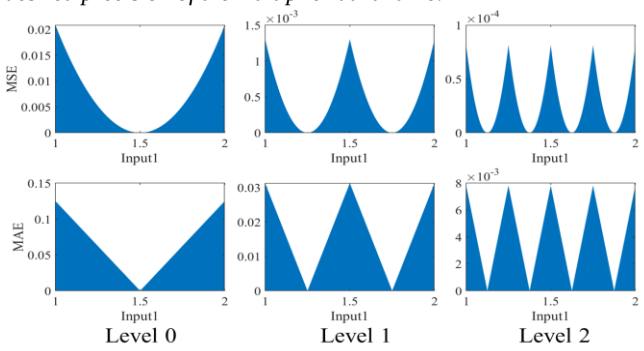

Figure 8: Error distribution for different error measures.

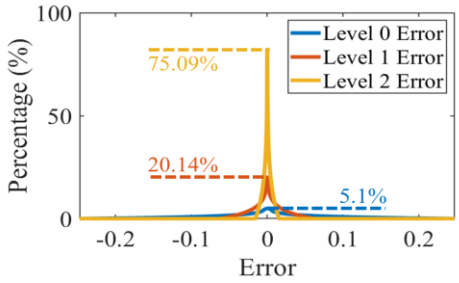

Figure 9: Envelopes of error distribution histogram for different leve ls of approximation for the proposed multiplier.

5 Experimental Results

We developed both software and hardware implementations of the proposed approximate multiplier for different evaluations. Similar as [26, 27, 30], the software implementation can be deployed into various applications by replacing the existing FP multiplier unit with the proposed approximated multiplier. The hardware implementation is on FPGA and we use Xilinx Vivado to evaluate the delay and energy. The experiments are organized as follows. We first qualitatively compare the proposed multiplier with several prior approximate multipliers. Then, we evaluate the performance of hardware implementation of the proposed multiplier. Since

ApproxLP [30] was reported to have SOTA performance in almost all the aspects when compared with the prior multipliers, in this paper, we directly compare with ApproxLP [30] and use the same setup for fairness. Finally, we evaluate the software performance in terms of quality and energy efficiency for various edge applications when using the proposed multiplier.

5.1 Qualitative Comparison to Prior Work

With the theoretically sound derivations and formulations in Sec. 3, we can design an optimally approximated and unbiased FP multiplier with runtime configurability. Here, we qualitatively compare the properties of the proposed multiplier with several prior approximate multiplier works in terms of unbiasedness (denoted as *biased*), configurability (denoted as *conf.*), request of full precision multiplier (denoted as *ReqMul*), and area complexity *w.r.t.* precision requirements (denoted as Complexity). Such comparison helps us better understand the superiority of the proposed multiplier in various aspects.

In Table 1, other than the first two multipliers [12, 28], the others are for FP multiplications. Note that RMAC [26] and CFPU [27] employ a hybrid solution to include both full precision multiplier and approximate multiplier, which is very area consuming. Thus, it is not fair to evaluate their area complexity for different precision requirements. ApproxLP [30], as SOTA in prior work, cannot guarantee unbiasedness and demand significant area with growing precision requirements. Thus, through comparison, we find the proposed multiplier can achieve good accuracy, unbiasedness, configurability, and low area complexity, which are the features highly desired by edge IoT applications.

Table 1: Qualitative comparison of the proposed multiplier with several prior approximate multipliers

Design	Type	Biased	conf.	ReqMul	Complexity
Kulkarni [12]	Integer	Y	N	N	$O(n^2)$
DRUM [28]	Fixed Pt	N	N	N	$O(n^2)$
CFPU [27]	FP	N	Y	Y	N/A
RMAC [26]	FP	Y	Y	Y	N/A
ApproxLP [30]	FP	Y	Y	N	$O(4^n)$
Proposed	FP	N	Y	N	$O(n)$

5.2 Quantitative Comparison to ApproxLP

Although both ApproxLP [30] and the proposed multiplier can employ multi-level approximations for configurability, there are some intrinsic differences in the multi-level partitioning, modeling, and implementation. Table 2 summarizes the number of sub-regions, LUT usage, and maximum delay for different levels of approximation for the two multipliers. Note that, with the growing approximation levels, the multiplier is expected to have higher precision and more sub-regions. For ApproxLP, the partitioning starts with 2 sub-regions (denoted as Level 1 in [30]), while the proposed design starts with 1 sub-region (no partitioning, denoted as Level 0). Here, for comparison purpose, ApproxLP [30] also starts with Level 0 (which is actually Level 1 in [30]). For ApproxLP, we report both its absolute number and relative change *w.r.t.* the proposed design using the same level of approximations.

As in Table 2, while the number of sub-regions for the proposed design grows faster than ApproxLP [30], its LUT usage is much lower, up to 84% less LUT usage for Level 2 approximation. For the maximum delay, due to the capability of parallel execution and lower hardware cost, the critical path for the proposed design has pretty much the same delay even with growing approximation levels. In contrast, ApproxLP shows 43% higher delay for Level 2 approximation when compared to the proposed design.

Table 2: Comparison of sub-region count, LUT usage, maximum delay between the proposed multiplier and ApproxLP [30].

Level	#sub-region		#LUT		Max delay (ns)	
	Prop.	ApproxLP	Prop.	ApproxLP	Prop.	ApproxLP
0	1	2	57	50/-12%	7.9	9.2/16%
1	4	4	95	160/68%	8.3	9.2/11%
2	16	12	151	278/84%	8.3	11.9/43%
3	64	N/A[1]	195	N/A	8.3	N/A

Table 3: Accuracy comparison for different error measures between the proposed multiplier and ApproxLP [30] for different approximation levels (Level 1 and 2).

Metrics	Level 1 approximation		Level 2 approximation	
	ApproxLP	Prop./Imp.	ApproxLP	Prop./Imp.
MSE	6.9e-4	4.3e-4/37%	4.3e-5	2.7e-5/37%
MAE	2.1e-2	1.6e-2/24%	5.2e-3	3.9e-3/25%
MSE(R)	1.7e-4	1.1e-4/35%	1.1e-5	6.9e-6/37%
MAE(R)	9.9e-3	7.6e-3/23%	2.5e-3	1.9e-3/24%

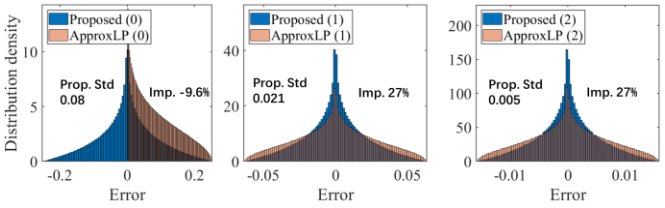

Figure 10: Error distribution comparison between the proposed multiplier and ApproxLP [30] for different approximation levels (0-2).

We further compare the accuracy of the proposed design *w.r.t.* ApproxLP [30] for different error measures and different approximation levels in Table 3. In the table, we have four different error measures, where MSE(R) and MAE(R) refer to the relative error for MSE and MAE, respectively. Note that ApproxLP [30] and the proposed design have different approximation schemes with different sub-regions for the same level of approximation. According to the same comparison principle in Table 2, 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. 10 shows the error distribution of the proposed design and ApproxLP. The proposed design can achieve much tighter error distribution with smaller standard deviation than ApproxLP, indicating consistently more accurate approximation.

[1] ApproxLP in [30] only reports the approximations up to 3 levels. Thus, 4_{th} or high level approximation data are not available.

Table 4: Comparison of PSNR/accuracy loss, energy and EDP improvements for different OpenCL, image processing and machine learning tasks using the proposed approximate multiplier with different levels of approximations (Level 0-2).

Application	PSNR(dB)/Accuracy Loss(%)			Energy Improvement			EDP Improvement		
	Level 0	Level 1	Level 2	Level 0	Level 1	Level 2	Level 0	Level 1	Level 2
Sobel	30.13	40.05	49.92	2.2×	2.2×	1.8×	3.5×	3.0×	2.4×
Kirsch	36.67	47.69	53.67	3.3×	2.6×	2.2×	5.2×	3.6×	2.9×
Robert	27.96	38.71	48.00	2.7×	2.0×	1.3×	4.2×	2.8×	1.8×
Prewitt	30.54	40.49	50.35	2.2×	2.2×	1.8×	3.5×	3.0×	2.4×
Gauss Blur	49.69	56.68	64.55	86.0×	54.7×	50.2×	136.3×	75.3×	66.1×
MNIST(MLP)	0.11%	0.02%	0%	62.6×	45.6×	41.8×	99.3×	62.7×	55.0×
MNIST(CNN)	0.02%	0%	0%	62.9×	50.3×	41.9×	99.7×	69.2×	55.2×
CIFAR-10	0.29%	0.01%	0%	121.9×	88.6×	81.3×	193.2×	122.0×	107.0×

5.3 Application Level Quality

We evaluate the efficiency of the proposed multiplier on several OpenCL, image processing and machine learning applications. MNIST(MLP) refers to the execution on a 5-layer MLP, while MNIST(CNN) and CIFAR-10 refer to the execution on a pre-trained AlexNet. Please note that in all the applications, more than 85% FP operations involve multiplications [27]. The embedded approximate multiplier can be configured to 3 levels of approximation (Level 0 to 2). The comparison is conducted by replacing the accurate FP multiplier in GPU with the proposed multiplier, as in [26, 27, 30]. The energy consumptions of the multipliers used in this section are obtained from Vivado. OpenCL/image processing tasks are evaluated with Peak Signal to Noise Ratio (PSNR), while machine learning tasks use accuracy loss as the accuracy measure.

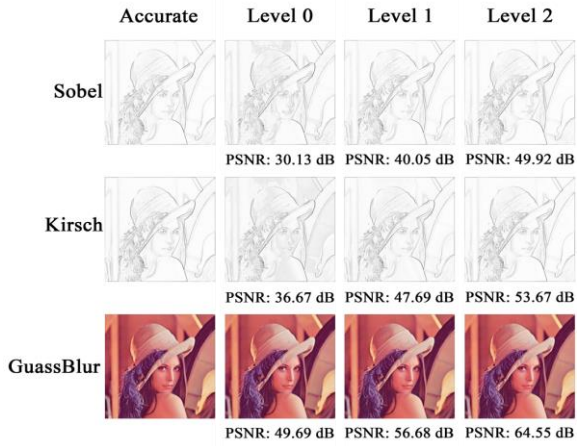

Figure 11: Output quality for image processing tasks using accurate and approximate multipliers with different approximate levels.

Table 4 reports the PSNR and accuracy loss for all the task. For OpenCL/image processing tasks, output PSNR is closely related to the distribution of the filter matrix. For example, Robert, Sobel and Prewitt filters have many 1's and 2's, which are located close to the corners of the subregions, while Kirsch filter has many 0's, 3's and 5's, located in the center of the sub-region. This explains why Kirsch has a higher PSNR than Robert or Sobel. Machine learning tasks have a strong resilience to error with consistently small accuracy loss. From the table, even with Level 1 approximation, we can achieve 38-57dB for all the OpenCL/image processing tasks and almost negligible accuracy loss (0-0.02%) for machine learning tasks. Fig. 11 presents an example of computation quality using Sobel, Kirsch and Gauss Blur running on the proposed multiplier with different approximation levels as well as the exact image (denoted as 'Accurate' on the first column). It is clear, the visual differences among the images are unnoticeable, while more approximation levels help increase the PSNR.

5.4 Application Level Efficiency

Table 4 also compares the normalized energy and energy-delay-product (EDP) improvements by the proposed multiplier in comparison to the case of using a full-precision FP multiplier. For tasks like Sobel, Kirsch, Robert and Prewitt, at least one input to the multiplier is relatively simple or stays stable during task execution. This simply implies there are fewer non-zero bits in an FP number, resulting in fewer bit switching and energy consumption during the task execution. Thus, for those tasks, EDP improvements by the proposed multiplier are limited, with only 4.1×, 3.1× and 2.4× on average for level 0, 1 and 2 approximations, respectively. For the tasks like Gauss Blur and machine learning, they have more varying inputs and many non-zeros in the FP numbers. This significantly increase the bit switching activity and hence the dynamic 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 193.2×, 122.0× and 107.0×, respectively.

6 Conclusions

In this work, an efficient runtime-configurable approximate multiplier is proposed. The multi-level architecture can easily incorporate the run-time configurability without incurring much area overhead, but naturally reach optimal approximation and unbiased error distribution. Our evaluation shows 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. The evaluations also demonstrate its significant energy efficiency improvement over full-precision multipliers in GPU.

Acknowledgement

This work was supported in part by National Key R&D Program of China (Grant No. 2018YFE0126300), the NSFC (Grant No. 61974133), and State Key Laboratory of ASIC & System (Grant No. 2020KF008).

REFERENCES

[1] C. Ji, Y. Li, W. Qiu, U. Awada and K. Li, "Big Data Processing in Cloud Computing Environments," *International Symposium on Pervasive Systems, Algorithms and Networks (ISPSAN)*, pp. 17-23, 2012.

[2] N. Khoshavi, X. Chen, J. Wang and R. F. DeMara, "Read-Tuned STT-RAM and eDRAM Cache Hierarchies for Throughput and Energy Enhancement," *arXiv preprint arXiv*:1607.08086, 2016.

[3] C. Zhuo, S. Luo, H. Gan, J. Hu and Z. Shi, "Noise-Aware DVFS for Efficient Transitions on Battery-Powered IoT Devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 39, no. 7, pp. 1498-1510, 2020.

[4] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347-2376, 2015.

[5] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni and J. Henkel, "Invited: Cross-layer Approximate Computing: From Logic to Architectures," *Design Automation Conference (DAC)*, pp. 1-6, 2016.

[6] M. Imani, M. Samragh, Y. Kim, S. Gupta, F. Koushanfar and T. Rosing, "RAPIDNN: In-Memory Deep Neural Network Acceleration Framework," *arXiv preprint arXiv*:1806.05794, 2018.

[7] M. Courbariaux, Y. Bengio and J. David, " Training Deep Neural Networks with Low Precision Multiplications," *arXiv preprint arXiv*:1412.7024, 2014.

[8] A. Suhre, F. Keskin, T. Ersahin, R. Cetin-Atalay, R. Ansari and A. E. Cetin, "A Multiplication-Free Framework for Signal Processing and Applications in Biomedical Image Analysis," *International Conference on Acoustics, Speech and Signal Processing (ICASSP),* pp. 1123-1127, 2013.

[9] M. Imani, S. Patil and T. S. Rosing, "MASC: Ultra-Low Energy Multiple-Access Single-Charge TCAM for Approximate Computing," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 373-378, 2016.

[10] K. He, A. Gerstlauer and M. Orshansky, "Circuit-Level Timing-Error Acceptance for Design of Energy-Efficient DCT/IDCT-Based Systems," *IEEE Transactions on Circuits and Systems for Video Technology (TCASVT)*, vol. 23, no. 6, pp. 961-974, 2013.

[11] S. Vahdat, M. Kamal, A. Afzali-Kusha, M. Pedram and Z. Navabi, "TruncApp: A Truncation-Based Approximate Divider for Energy Efficient DSP Applications," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1635-1638, 2017.

[12] P. Kulkarni, P. Gupta and M. Ercegovac, "Trading Accuracy for Power with an Underdesigned Multiplier Architecture," *International Conference on VLSI Design (ICVLSI)*, pp. 346-351, 2011.

[13] M. Imani, K. Yeseong and R. Tajana, "Acam: Approximate Computing Based on Adaptive Associative Memory with Online Learning," *International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 162-167, 2016.

[14] C. Zhuo, K. Unda, Y. Shi, and W. Shih, "From Layout to System: Early Stage Power Delivery and Architecture Co-Exploration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 38, issue 7, pp. 1291-1304, 2019.

[15] J. Deng, Z. Shi, and C. Zhuo, "Energy Efficient Real-Time UAV Object Detection on Embedded Platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 10.1109/TCAD.2019.2957724, 2019.

[16] J. Han and M. Orshansky, "Approximate Computing: An Emerging Paradigm for Energy-Efficient Design," *European Test Symposium (ETS)*, pp. 1-6, 2013.

[17] M. Imani, A. Rahimi and T. S. Rosing, "Resistive Configurable Associative Memory for Approximate Computing," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1327-1332, 2016.

[18] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy and A. Raghunathan, "Quality Programmable Vector Processors for Approximate Computing," *International Symposium on Microarchitecture (MICRO)*, pp. 1-12, 2013.

[19] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan and K. Roy, "IMPACT: IMPrecise Adders for Low-Power Approximate Computing," *International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 409-414, 2011.

[20] M. S. Razlighi, M. Imani, F. Koushanfar and T. Rosing, "LookNN: Neural Network with No Multiplication," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1775-1780, 2017.

[21] M. Imani, M. Masich, D. Peroni, P. Wang and T. Rosing, "CANNA: Neural Network Acceleration Using Configurable Approximation on GPGPU," *Asia and South Pacific Design Automation Conference (ASPDAC)*, pp. 682-689, 2018.

[22] Jian Liang, R. Tessier and O. Mencer, "Floating Point Unit Generation and Evaluation for FPGAs," *International Symposium on Field-Programmable Custom Computing Machines (ISFPCCM)*, pp. 185-194, 2003.

[23] R. K. Yu and G. B. Zyner, "167 MHz Radix-4 Floating Point Multiplier," *International Symposium on Computer Arithmetic (ISCA)*, pp. 149-154, 1995.

[24] Gokul Govindu, L. Zhuo, S. Choi and V. Prasanna, "Analysis of High-Performance Floating-Point Arithmetic on FPGAs," *International Parallel and Distributed Processing Symposium (IPDPS),* pp. 149-, 2004.

[25] 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," *European Solid-State Circuits Conference (ESSCIRC)*, pp. 465-468, 2016.

[26] M. Imani, R. Garcia, S. Gupta, and T. Rosing, "RMAC: Runtime Configurable Floating Point Multiplier for Approximate Computing," *International Symposium on Low Power Electronics and Design (ISLPED)*, no. 12, pp. 1–6, 2018.

[27] M. Imani, D. Peroni and T. Rosing, "CFPU: Configurable Floating Point Multiplier for Energy-Efficient Computing," *Design Automation Conference (DAC)*, pp. 1-6, 2017.

[28] S. Hashemi, R. I. Bahar and S. Reda, "DRUM: A Dynamic Range Unbiased Multiplier for Approximate Applications," *International Conference on Computer-Aided Design (ICCAD)*, pp. 418-425, 2015.

[29] 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 (TVLSI) Systems*, vol. 23, no. 6, pp. 1180-1184, 2015.

[30] M. Imani et al., "ApproxLP: Approximate Multiplication with Linearization and Iterative Error Control," *Design Automation Conference (DAC)*, pp. 1-6, 2019.

[31] IEEE Standard for Floating-Point Arithmetic, *IEEE standard 754-2008*, pp. 1-70, 2008.

[32] C. Liu, J. Han and F. Lombardi, "A Low-Power, High-Performance Approximate Multiplier with Configurable Partial Error Recovery," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1-4, 2014.

[33] K. Bhardwaj, P. S. Mane and J. Henkel, "Power- and Area-Efficient Approximate Wallace Tree Multiplier for Error-Resilient Systems," *International Symposium on Quality Electronic Design (ISQED)*, pp. 263-269, 2014.

[34] C. Lin and I. Lin, "High Accuracy Approximate Multiplier with Error Correction," *International Conference on Computer Design (ICCD)*, pp. 33-38, 2013.

[35] S. E. Ahmed, S. Kadam and M. B. Srinivas, "An Iterative Logarithmic Multiplier with Improved Precision," *International Symposium on Computer Arithmetic (ARITH)*, pp. 104-111, 2016.