A Survey on Approximate Multiplier Designs for Energy Efficiency: From Algorithms to Circuits

YING WU, Zhejiang University, China
CHUANGTAO CHEN, Zhejiang University, China
WEIHUA XIAO, Shanghai Jiao Tong University, China
XUAN WANG, Shanghai Jiao Tong University, China
CHENYI WEN, Zhejiang University, China
JIE HAN, University of Alberta, Canada
XUNZHAO YIN, Zhejiang University, China
WEIKANG QIAN, Shanghai Jiao Tong University, China
CHENG ZHUO, Zhejiang University, China

Given the stringent requirements of energy efficiency for Internet-of-Things edge devices, approximate multipliers have recently received growing attention, especially in error-resilient applications. The computation error and energy efficiency largely depend on how and where the approximation is introduced into a design. Thus, this article aims to provide a comprehensive review of the approximation techniques in multiplier designs ranging from algorithms and architectures to circuits. We have implemented representative approximate multiplier designs in each category to understand the impact of the design techniques on accuracy and efficiency. The designs can then be effectively deployed in high level applications, such as machine learning, to gain energy efficiency at the cost of slight accuracy loss.

CCS Concepts: • **General and reference → Surveys and overviews**; • **Hardware → Arithmetic and datapath circuits**; *System-level fault tolerance.*

Additional Key Words and Phrases: approximate computing, multiplier, algorithm, architecture, circuit

ACM Reference Format:
Ying Wu, Chuangtao Chen, Weihua Xiao, Xuan Wang, Chenyi Wen, Jie Han, Xunzhao Yin, Weikang Qian, and Cheng Zhuo. 2023. A Survey on Approximate Multiplier Designs for Energy Efficiency: From Algorithms to Circuits. 1, 1 (November 2023), 36 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The rapid growth of Internet-of-Things (IoT) has made energy efficiency a critical concern for IoT devices due to the constrained resources on the edge [1]. Conventional processors, such as the central processing unit (CPU) and the graphics processing unit (GPU), compute with pre-determined

Authors' addresses: Ying Wu, Zhejiang University, Hangzhou, China, ying.wu@zju.edu.cn; Chuangtao Chen, Zhejiang University, Hangzhou, China, chtchen@zju.edu.cn; Weihua Xiao, Shanghai Jiao Tong University, Shanghai, China, 019370910014@sjtu.edu.cn; Xuan Wang, Shanghai Jiao Tong University, Shanghai, China, xuan.wang@sjtu.edu.cn; Chenyi Wen, Zhejiang University, Hangzhou, China, wwency@zju.edu.cn; Jie Han, University of Alberta, Edmonton, Alberta, Canada, jhan8@ualberta.ca; Xunzhao Yin, Zhejiang University, Hangzhou, China, xzyin1@zju.edu.cn; Weikang Qian, Shanghai Jiao Tong University, Shanghai, China, qianwk@sjtu.edu.cn; Cheng Zhuo, Zhejiang University, Hangzhou, China, czhuo@zju.edu.cn.

XXXX-XXXX/2023/11-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

but unnecessary full precisions for all the computational tasks, which inevitably reduces the energy efficiency. However, for many error-tolerant applications, such as neural networks, computational approximation or inaccuracy can be tolerated without impacting the inference accuracy [2]. Thus, it is highly desired to optimize the energy efficiency for resource-constrained IoT devices by providing sufficient instead of excessively accurate outputs.

Given the error tolerance nature of many IoT applications [3–7], approximate computing that trades numerical precision for computational efficiency has become a promising alternative to achieving the needed energy efficiency with quality results while consuming minimum resources. For computationally intensive applications, *e.g.*, neural networks and image processing, **multiplication** is probably the most frequently invoked operation [8], which requires significant energy and a long latency to provide accurate outputs.

As a common arithmetic component, the multiplier has been studied for decades [9, 10]. The focus was mainly on accuracy and performance in fully precise computations. Mitchell proposed an approximate logarithm-based multiplier in the 1960s [11], which, however, failed to attract significant attention for decades. Since then, the approximation technique of truncation and its variants, such as the fixed-width multiplier that enforces the same bit-width for both inputs and outputs, have been proposed for achieving area efficiency [12, 13]. To reduce the truncation induced inaccuracy, around 2000, a few compensation techniques were suggested at the cost of additional complexity [14–18]. Recently, with the popularity of IoT devices and their stringent resource constraints, researchers from both academia and industry have devoted significant efforts to approximate multiplier design and optimization, which resulted in many effective and interesting approximation techniques [8, 19–38]. In general, such efforts can be categorized at multiple levels, ranging from algorithms, architectures, to circuits:[1]

- At the algorithm level, the Mitchell's multiplier has been further improved to reduce the bias or hardware costs [39–42]. Additionally, a linear-fitting algorithm has been innovatively applied to transform multiplication to simpler operations such as addition and shift [31, 32].
- Approximate techniques at the architecture level are more frequently revisited and explored. Various new techniques have been introduced at different stages of a binary multiplication following the conventional data flow [24, 36, 43–47, 47–56].
- At the circuit level, various approximation techniques have been incorporated into the low-level implementations of the sub-modules of a multiplier [24, 51, 54, 55, 55, 57, 58]. Such circuit-level techniques can be integrated into the designs at the other two levels.

Approximate multipliers have already been successfully applied in error-tolerant tasks, such as filtering, image processing, and machine learning [14, 15, 37–39, 59–72]. In 1985, Ashtaputre *et al.* designed a systolic array with approximate multipliers and demonstrated that the inaccurate multiplication can bring negligible impact on the results [59]. Around 2000, fixed-width multiplier [12, 13] were widely used in digital signal processing such as filtering [15] and wavelet transformation [14]. Snigdha *et al.* applied approximate multipliers in image compression blocks and observed over 12% improvements in area, power, and delay [64]. Hammad *et al.* replaced the original full-precision multipliers in VGG networks with approximate ones to support the classification tasks on CIFAR-10 and CIFAR-100, which again showed negligible accuracy losses [68]. After that, they further proposed to deploy approximate multipliers in training to improve the performance [70].

Scope of the article: Since many prior designs rely on hand-crafted structures or heuristics, it is then highly desired to systematically review and understand the advantages and disadvantages of the various alternatives to introducing approximation into a design. Hence, this article provides

[1]The algorithm in this article means the computation procedure of multiplication rather than a problem-solving process. A more clear definition of the three levels is given in Section 3.

a comprehensive review of the approximate multipliers, including approximation techniques at the algorithm, architecture, and circuit levels, the comparison of different techniques using various metrics, and the discussion of the future trend. It is noted that we re-implement various approximation techniques in Verilog to ensure the fair comparison in our evaluation, which will be released as an open-source library, **AM-Lib** [73].

The remainder of this article is organized as follows. In Section 2, we review the background of approximate multipliers. Section 3 provides an overview of the approximate multipliers. Sections 4 to 6 discuss the approximate multipliers at the algorithm, architecture, and circuit levels, respectively. Note that some presented designs utilize approximation techniques at multiple levels. In Section 7, we evaluate the accuracy and hardware cost of approximate multipliers and their applications in neural networks by considering accuracy losses and energy efficiency. Then, a discussion is presented in Section 8 followed by the conclusions in Section 9.

2 BACKGROUND

2.1 Fixed-point vs. Floating-point Representations

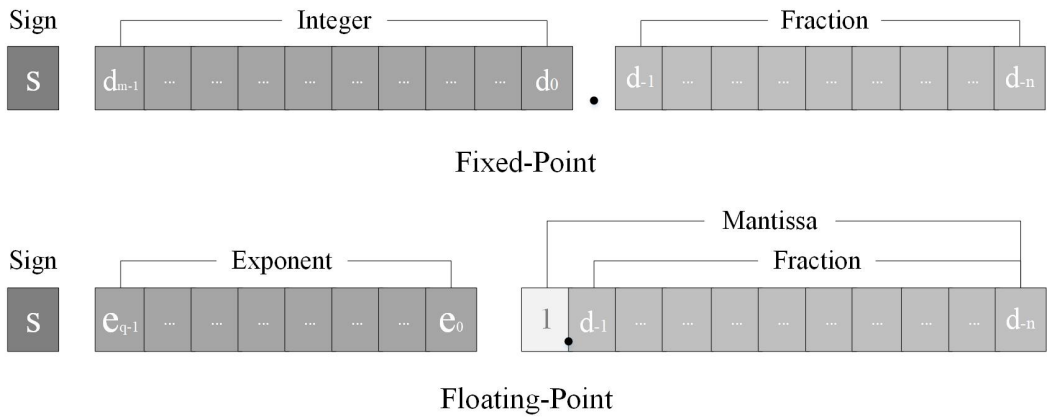

Fig. 1. Comparison between fixed-point and floating-point formats.

Similar as many arithmetic functions implemented in hardware, the multiplier design can be categorized to fixed-point and floating-point implementations as a trade-off among accuracy, dynamic range, and cost. The major difference between fixed-point and floating-point numbers lies in whether the implementation has a specific number of digits reserved for the integer and fractional parts. In other words, a fixed-point representation uses a decimal point at a fixed position. Obviously, a floating-point representation may offer a wider dynamic range and higher precision than its fixed-point counterpart, but at the cost of area, speed, and power consumption.

Fig. 1 compares the fixed-point and the floating-point formats in the binary number system. The fixed-point format consists of a sign bit, an integer part, and a fractional part, with a fixed binary point position. On the other hand, according to the IEEE 754 standard [74], which is a technical standard for floating-point arithmetic, a floating-point number consists of a sign, an exponent, and a mantissa. The mantissa of a *normalized* floating-point number is a fraction between 1 and 2, where its first digit is fixed to 1 and the rest is the fraction in the range of [0,1).

With the underlying number representations, designers may use either a fixed-point or a floating-point multiplier to conduct multiplication. For the floating-point numbers, the multiplication

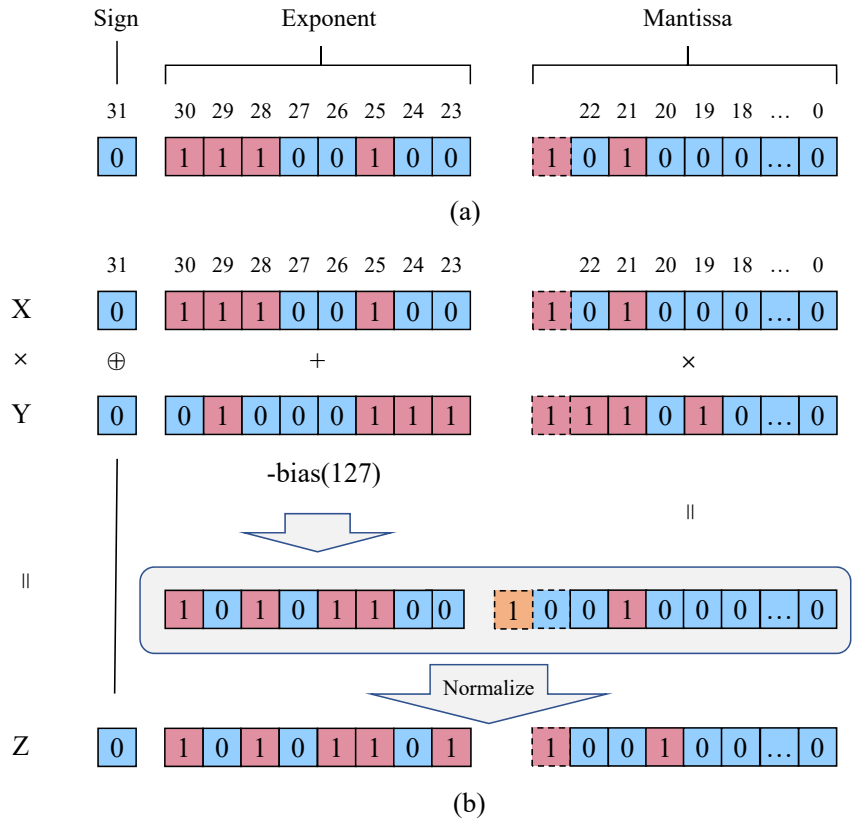

Fig. 2. An example of a 32-bit floating-point multiplication according to IEEE 754 standard.

procedure of 32-bit floating-point numbers is shown in Fig. 2. The sign bits are XORed and the exponents are summed by an adder. Then, a bias of 127 is subtracted from the sum to allow both negative and positive values for the exponent. Finally, the two mantissas are multiplied and shifted to the range of 1 and 2 to produce the normalized representation. The exponent will be adjusted if a shift happens. For a floating-point multiplication, the computation on the mantissa part requires more energy and a longer delay than the other two parts, which is hence the focus of most research work [31, 32]. On the other hand, if no overflow occurs, the fixed-point multiplication is carried out as a regular multiplication with its fractional part truncated to the specified bit-width. However, the difference between the two multiplications is actually smaller than it seems. The multiplication of the mantissa parts of floating-point numbers can be always viewed as a special case of fixed-point multiplication, where the integer part is 1. Thus, the most critical operations in fixed- and floating-point multiplications can be considered as the same.

2.2 Binary Multipliers

Before we elaborate further the approximate multiplier design, we introduce the basics of a binary multiplier in this section, which sheds light on different approximation techniques introduced in the latter sections.

Due to the nature of dealing with the two digits, *i.e.*, 0 and 1, the binary multiplication can be considered as a process of addition and shifting. For example, consider two 4-bit operands

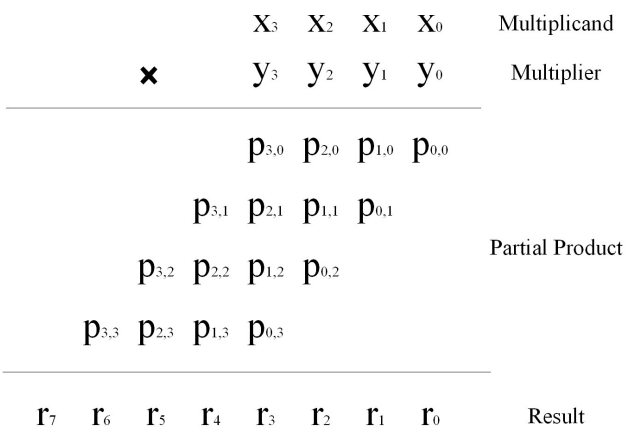

Fig. 3. An example of 4-bit multiplication.

of x and y, where x is the multiplicand and y is the multiplier. As shown in Fig. 3, similar to decimal multiplication, the binary multiplication is carried out for each bit of the multiplier (*i.e.*, y_i for $i = 0, 1, 2, 3$) and the multiplicand (*i.e.*, x=$\{x_3, x_2, x_1, x_0\}$) to generate a partial product, *e.g.*, $\{p_{3,0}, p_{2,0}, p_{1,0}, p_{0,0}\}$ for the first row. This process is then repeated for each bit of the multiplier y, with the partial product left-shifted by 1 bit. Finally, all the partial products are accumulated to obtain the final product as $\{r_7, r_6, r_5, r_4, r_3, r_2, r_1, r_0\}$. Thus, the operation of a binary multiplier can be roughly divided into three stages, data input, partial product generation, and accumulation.

Since the partial product is generated without a carry, the bit-wise multiplication can be calculated with AND gates. Once all the partial products are generated, we can use an array of adders to accumulate the partial products as shown in Fig. 4, where HA refers to a half adder and FA refers to a full adder. Obviously, the critical path of such a structure is the carry propagation chain. For example, Fig. 4(a) shows a ripple-carry adder (RCA)-based accumulator, with the carries propagated horizontally from right to left, while Fig. 4(b) shows a carry-save adder (CSA)-based accumulator with carries propagated diagonally to achieve a shorter critical path for faster speed.

In order to further accelerate the accumulation, Wallace proposed a tree structure in 1964 [75]. As shown in Fig. 5, the Wallace Tree groups three partial products within one column to generate two outputs, *i.e.*, a sum and a carry, thereby reducing the number of partial products by a factor of approximately 1.5. The operation is repeated until only two rows are left, *e.g.*, by the first 3 steps in Fig. 5. Then, the last two rows are added up to obtain the final result. Parallel computation and partial product compression in each stage can be utilized to speed up the accumulation process [75].

2.3 Exact Compressor

In the tree-based partial product accumulation, such as Wallace tree [75] and Dadda tree [76], compressors are used to count the number of ones within a group of partial products. The full adder and half adder in Fig. 5 are widely used as 3-2 compressor and 2-2 compressor, respectively. In order to further improve the compression efficiency, higher-order compressors, such as 4-2 and 5-2 compressors [77–81] or 6-3 and 7-3 compressors [82], have also been investigated. Fig. 6 shows the design of a conventional 4-2 compressor. It consists of two full-adders and has five primary inputs (PIs), x_1, x_2, x_3, x_4, and C_{in}, and three primary outputs (POs), *Sum*, *Carry*, and C_{out}, where C_{in} comes from the preceding less significant block, and C_{out} goes to the next more significant block.

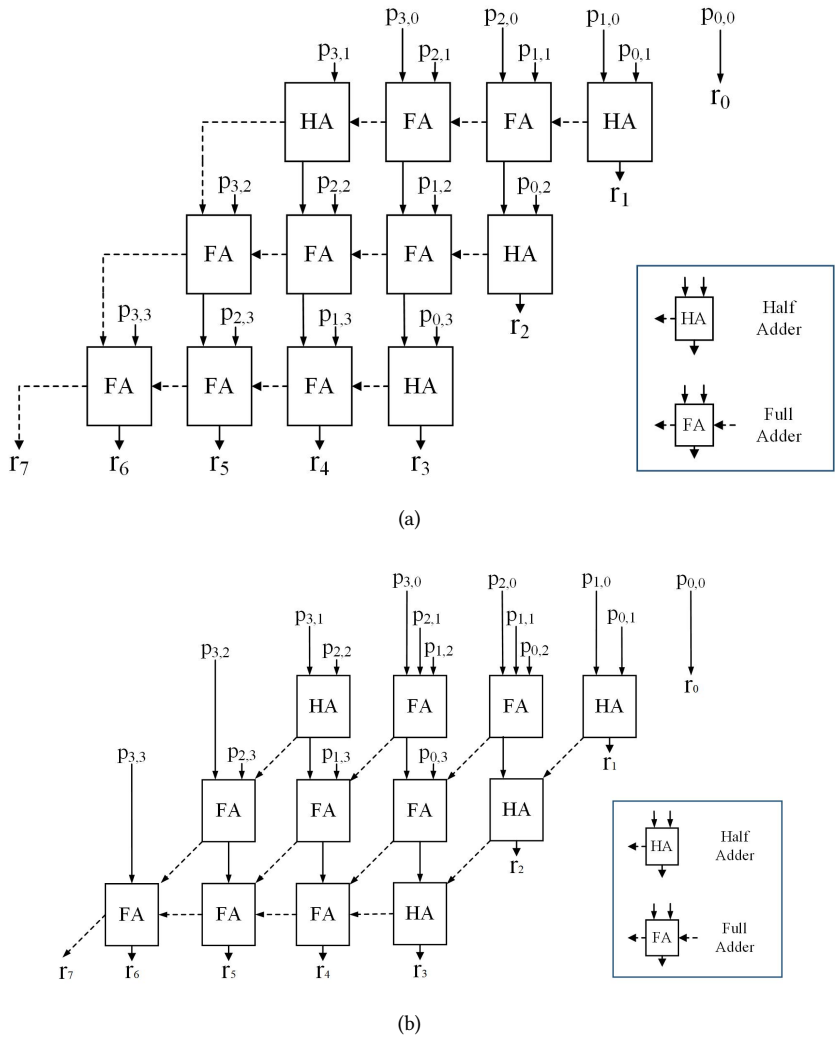

Fig. 4. (a) An example of RCA-based accumulator; (b) An example of CSA-based accumulator.

The outputs *Sum*, *Carry*, and C_{out} can be calculated as [83]:

$$Sum = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus C_{in},$$
$$Carry = (x_1 \oplus x_2 \oplus x_3 \oplus x_4) \wedge C_{in} + \overline{(x_1 \oplus x_2 \oplus x_3 \oplus x_4)} \wedge x_4,$$
$$C_{out} = (x_1 \wedge (x_2 \vee x_3)) \vee (x_2 \wedge x_3).$$

Note that the bit significances of the *Carry* and C_{out} are both twice of that of *Sum*.

3 OVERVIEW OF APPROXIMATE MULTIPLIERS

Many prior works on approximate multiplier have tackled the problem by introducing approximations at algorithm, architecture, or circuit levels to reduce the critical path delay or improve the

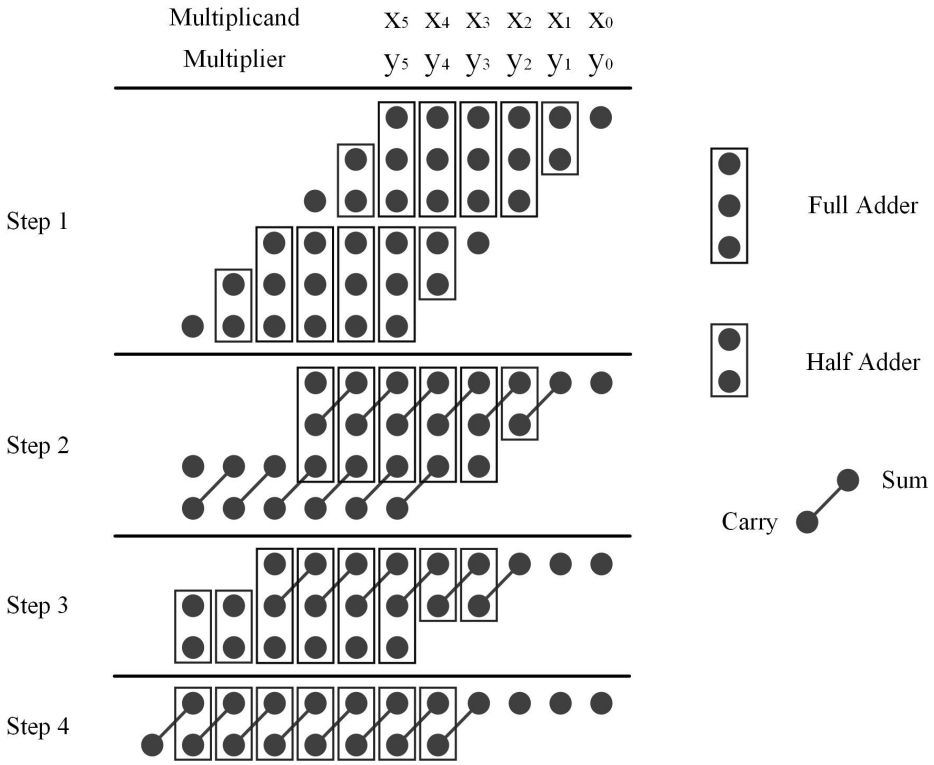

Fig. 5. An example of Wallace Tree-based multiplier.

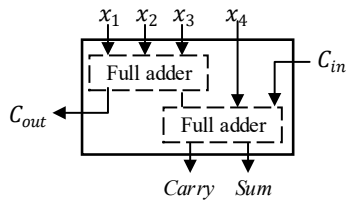

Fig. 6. A conventional exact 4-2 compressor.

energy efficiency. For example, at the algorithm level, Ahmed *et al.* explored a pipelined log-based approximation using the classical Mitchell's multiplier with an iterative procedure to improve the accuracy [84]. At the architecture level, many works focused on improving the conventional multiplier architecture with approximate components, such as adders, to speed up the addition or partial product generation [24, 45, 85–88]. At the circuit level, references [24, 51, 54, 55, 57, 58, 89–91] proposed to approximate Boolean algebra expressions or prune out a few gates to simplify the circuit.

For all the prior works with various approximation techniques, it is actually very challenging to precisely categorize the introduced approximation to a particular design level, *i.e.*, algorithm, architecture, or circuit levels. Many of them actually involve multiple design levels, as the highest level approximation, *i.e.*, the algorithm level, may incur additional architecture changes [31, 32, 84,

92, 93]. Similarly, circuit-level techniques are often utilized for basic module designs applied in higher-level approximation schemes [24, 30, 40, 46, 49–56, 58, 89, 94].

In order to facilitate our review, we introduce the following definitions to categorize the considered approximation:

- Algorithm: The introduced approximation originates from a different algorithmic procedure to conduct multiplication.
- Architecture: With the binary multiplier architecture in Section 2.2 as a reference, the introduced approximation is intended to improve the efficiency of a particular stage in the reference architecture.
- Circuit: The approximation technique is not limited to a particular architecture or algorithm, and it can be combined with the approximation techniques at other design levels.

Table 1 summarizes the state-of-the-art works reviewed in this article. These works are listed in the reverse chronological order. We mark the types of approximate multipliers (where symbols U, S, and FP indicate fixed-point unsigned, fixed-point signed, and floating-point multipliers, respectively) and the utilized techniques in the three considered levels. Due to the space limit, we use some abbreviations to represent the approximation techniques or stages in the remainder of this article:

- The abbreviations *log*, *linear*, and *hy* in the algorithm level indicate logarithm-based, linearization-based, and hybrid schemes, respectively.
- In the architecture level, the abbreviations *in*, *pp*, *acc*, and *enc* denote input, partial product, accumulation, and encoder, respectively, representing the four stages in a multiplier to introduce the approximation.
- The abbreviations *BR*, *GP*, *ECD*, and *VOS* in the circuit level denote Boolean rewriting, gate-level pruning, evolutionary circuit design, and voltage over-scaling techniques, respectively.

The techniques mentioned above will be described in details in what follows.

4 APPROXIMATE MULTIPLIER WITH ALGORITHM-LEVEL APPROXIMATION

The algorithm-level approximate multiplier rebuilds the multiplication algorithm itself, which naturally results in a new multiplier architecture. In this section, we will review three different multiplier approximations at the algorithm level: logarithm-based approximation, approximation with linearization, and hybrid approximation.

4.1 Logarithm-Based Approximation

With logarithmic transformation, the multiplication can be converted to addition, where the two operands are the logarithms of multiplicand and multiplier, respectively. The first logarithm-based multiplier (LM) was proposed by Mitchell in 1962 [11]. For a multiplication of two operands A and B, we have:

$$A = 2^{k_1}(1 + x_1) \,, \tag{1}$$

$$\log_2(A) = k_1 + \log_2(1 + x_1) \,, \tag{2}$$

where k_1 indicates the position of the leading one of A and x_1 is the fraction part of A that lies in $[0, 1)$. The same formulation can be applied to the other operand B with the parameters of k_2 and x_2. The logarithm of the product can be written as:

$$\log_2(A \times B) = k_1 + k_2 + \log_2(1 + x_1) + \log_2(1 + x_2) \,. \tag{3}$$

According to Eq. (3), the implementation based on Mitchell's algorithm [11] requires leading one detectors (LODs), binary-logarithm converters (BLCs), adders, and logarithm-binary converters

Table 1. Summary of the state-of-the-art designs of approximate multipliers.

Year	Work	Type			Algorithm			Architecture				Circuit			
		U	S	FP	log	linear	hy	in	pp	acc	enc	BR	ECD	GP	VOS
2022	[95]	✓								✓					
2020	[40]	✓			✓							✓			
2020	[32]			✓		✓									
2020	[50]	✓								✓		✓			
2020	[54]	✓								✓		✓			
2019	[31]			✓		✓									
2019	[36]	✓	✓					✓		✓					
2019	[52]	✓								✓		✓			
2019	[41]	✓			✓										
2018	[93]			✓			✓								
2018	[47]	✓							✓	✓					
2018	[51]	✓								✓		✓			
2018	[39]	✓			✓							✓			
2018	[42]	✓		✓	✓										
2018	[58]			✓						✓	✓	✓			
2018	[49]	✓								✓		✓			
2018	[94]	✓							✓	✓		✓			
2017	[46]	✓							✓	✓		✓			
2017	[92]			✓			✓								
2017	[44]	✓	✓					✓		✓					
2017	[48]														
2017	[55]		✓							✓	✓	✓			
2017	[96]												✓		
2016	[89]		✓							✓	✓	✓			
2016	[84]	✓			✓			✓							
2016	[97]												✓		
2015	[43]	✓						✓							
2015	[56]		✓								✓	✓			
2015	[53]	✓								✓		✓			
2015	[91]									✓				✓	
2014	[45]	✓						✓							
2013	[98]		✓							✓					
2012	[99]														✓
2011	[24]	✓							✓			✓			
2011	[100]		✓							✓					
2009	[101]	✓								✓					
2009	[102]	✓													✓
2009	[30]	✓													✓
2009	[103]		✓							✓					
2007	[104]		✓							✓					
2004	[17]		✓							✓		✓			
1962	[11]	✓			✓										

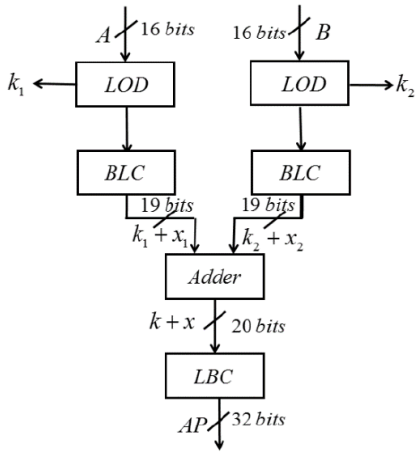

Fig. 7. Procedure for the Mitchell's algorithm [11].

(LBCs). The procedure for Mitchell's algorithm is illustrated in Fig. 7 for a 16×16 multiplier. To reduce the implementation complexity, the logarithmic expression in Eq. (3) is approximated as:

$$\log_2(x+1) \approx x, 0 \le x < 1 \ . \tag{4}$$

Then we have: $A \times B \approx 2^{k_1+k_2+x_1+x_2} = 2^{k_1+k_2} \times 2^{x_1+x_2}$. Based on the carry of $x_1 + x_2$, Eq. (4) can be further approximated as:

$$A \times B \approx \begin{cases} 2^{k_1+k_2}(x_1+x_2+1), & x_1+x_2 < 1 \ , \\ 2^{k_1+k_2+1}(x_1+x_2), & x_1+x_2 \ge 1 \ . \end{cases} \tag{5}$$

Compared with the original multiplication, when $x_1 + x_2 < 1$, the error of Eq. (5) is given by:

$$\begin{aligned} Error &= A \times B - 2^{k_1+k_2}(x_1+x_2+1) \\ &= 2^{k_1+k_2}(1+x_1)(1+x_2) - 2^{k_1+k_2}(x_1+x_2+1) \\ &= 2^{k_1+k_2}x_1x_2 \end{aligned} \tag{6}$$

It is noted that the error term of $2^{k_1+k_2}x_1x_2$ has the same structure as $A \times B = 2^{k_1+k_2}(1+x_1)(1+x_2)$. Then we can repeat the approximation procedure to compute $2^{k_1+k_2}x_1x_2$, which indicates an iterative process to achieve a higher accuracy using logarithm-based approximation. In [84], the iterative approximation for $x_1+x_2 \ge 1$ has been explored together with a truncation scheme. Liu *et al.* further investigated the logarithm-based approximate multipliers using different approximate adders and find that multipliers using set-one-adders (SOAs) can achieve a higher accuracy [39]. As shown in Fig. 8, an SOA consists of one approximate adder for the lower m bits and one exact adder for the higher $(n-m)$ bits. The approximate adder always sets the lower m sum bits to logic 1 and hence, results in over-estimation. Such an over-estimation is particularly designed to compensate for the accuracy loss of a logarithm-based approximate multiplier, as Mitchell's algorithm always underestimates the multiplication result. Similar compensation schemes have been introduced in [40–42] to improve the average error introduced by Mitchell's algorithm at the cost of area and power consumption. For example, Ansari *et al.* proposed an improved logarithm transformation algorithm, which introduces double-sided errors to reduce the error bias [40].

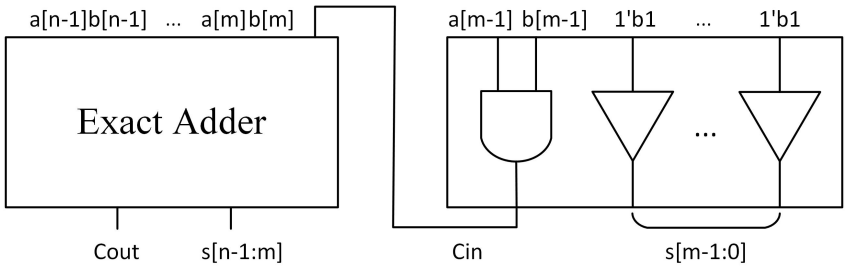

Fig. 8. Architecture of an n-bit set-one-adder [105].

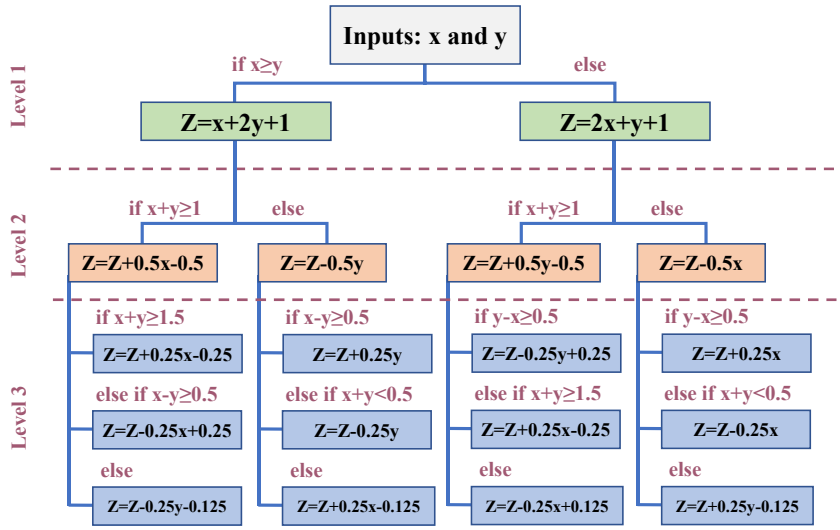

Fig. 9. Computation procedure using the linear approximation in [31].

4.2 Approximation with Linearization

Multiplication is a nonlinear operation that can be implemented with additions and compressions. In mathematics, it is a natural idea to approximate a nonlinear curve with a piece-wise linear function. Thus, researchers have attempted to use linear arithmetic operations to approximate the nonlinear multiplication [31, 32]. It is noted that, while logarithm-based approximate designs are based on Eq. (5), it is actually a special case of linearization.

Without loss of generality, the multiplication can be considered as a function of two variables, whose linear approximation can be expressed as:

$$f = xy \approx f_{approx} = ax + by + c;, \tag{7}$$

where x and y are the input operands, and a, b, and c are coefficients. In [31], an iterative linear approximation for floating-point multiplication is proposed to approximate the multiplication according to Eq. (7). For the mantissas of normalized floating-point numbers, the range of the product is $[1, 2) \times [1, 2)$, which is a square domain. By appropriately partitioning the domain into smaller sub-domains and assigning a proper linear function to each, the original nonlinear surface for the multiplication can be approximated by a series of piece-wise linear functions, one

for each sub-domain. Fig. 9 summarizes the computation procedure called ApproxLP using the linear approximation in [31]. It is clear that the accuracy can be improved by partitioning more sub-domains, the number of which grows exponentially with the approximation level. Thus, the efficiency of ApproxLP in [31] actually quickly degrades with a larger approximation level. Moreover, the comparators used for each level in Fig. 9 also introduce non-trivial delay overhead. Fig. 10 plots the error distributions of mantissa multiplications in ApproxLP for different approximation levels with x-axis and y-axis representing the mantissa range.

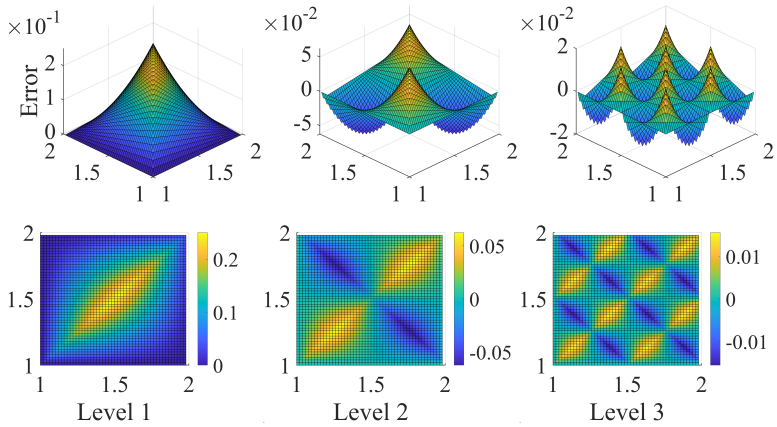

Fig. 10. Error distributions of ApproxLP at different approximation levels [31].

To reduce the number of comparators, Chen *et al.* proposed to partition the input domain into identical smaller square sub-domains [32]. For one level higher, each domain (or sub-domain) is further partitioned into four identical smaller ones. With such an iterative process, there are 4^n sub-domains for level-n approximation. For a rectangular domain $[x_1, x_2] \times [y_1, y_2]$, the optimal coefficients to minimize the mean square error (MSE) between $f_{approx} = ax + by + c$ and $f = xy$ are [32]:

$$a = \frac{y_1 + y_2}{2}, b = \frac{x_1 + x_2}{2}, c = -ab. \tag{8}$$

Fig. 11 demonstrates the multi-level approximate multiplier architecture of an optimally approximated multiplier (OAM) in [32]. In the figure, Level 0 is denoted as the basic approximation module, which 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 depth. Unlike ApproxLP [31], comparators are no longer needed for OAM [32]. Thus, the delay of OAM can be significantly reduced when compared to ApproxLP even for a similar number of sub-domains.

Since the two coefficients of a and b are the middle points of the intervals where the operands belong to, a circuit-friendly implementation can be achieved for the error compensation at each

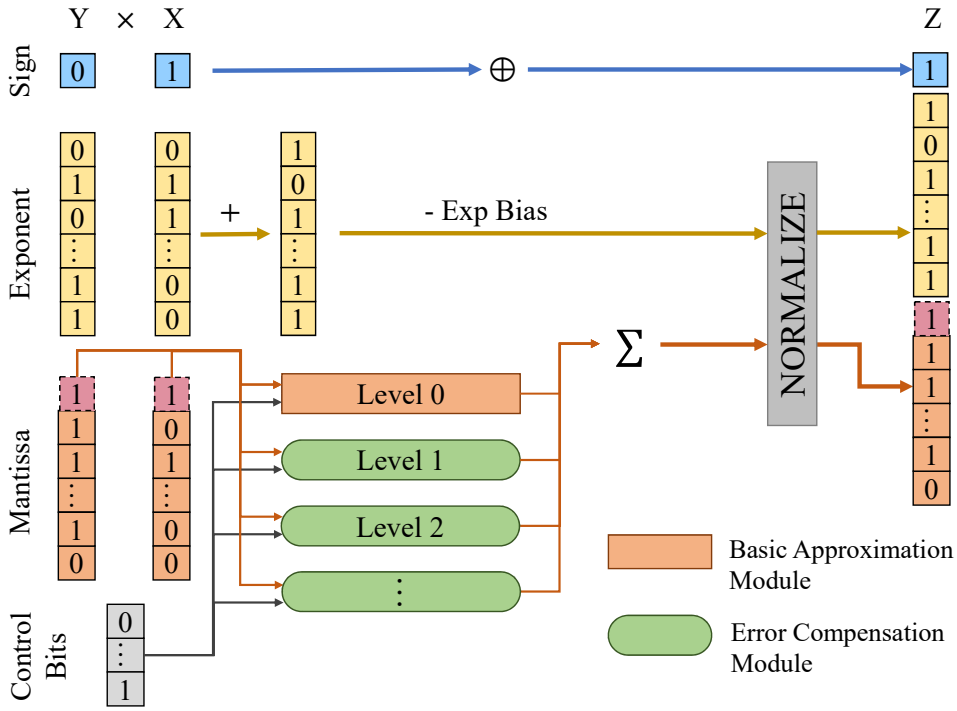

Fig. 11. Architecture of the approximate multiplier OAM in [32].

level as [32]:

$$\begin{aligned}\Delta f_n = &\Big\{\Big[\big(x[n]?(1):(-1)\big)\times\big(y-y_{n-1}^{\hat{}}\big)\Big]\\ &+\Big[\big(y[n]?(1):(-1)\big)\times\big(x-x_{n-1}^{\hat{}}\big)\Big]\Big\}\gg(n+1)\\ &+\Big[\big(x[n]\oplus y[n]\big)?(1):(-1)\Big]\gg(2n+2),\end{aligned}\tag{9}$$

where "? :" is the conditional operator, $\oplus$ is the XOR operator, $\gg$ is the right shift operator, $x[n]$ is the n^{th} most significant bit (MSB) of mantissa x, and $x_{n-1}^{\hat{}}$ preserves the $(n-1)$ MSBs of mantissa x with an extra bit 1 at the n^{th} MSB position. Since the number of right shifts is pre-determined at each level, the right shift operation does not require an additional circuit to implement. Thus, the number of operations at each level for the OAM is reduced to 5, including one XOR operation, two arithmetic negations, and two additions, which results in a constant area complexity, whereas ApproxLP has an area complexity of $O(4^n)$ [32]. In addition, a hardware-friendly implementation of the OAM is provided in [106].

The errors of the OAM [32] are shown below in terms of maximum error distance (MaxED), mean error distance (MED), and MSE for the approximation level n:

$$\text{MaxED}=\frac{1}{4^{n+1}},\ \text{MED}=\frac{1}{4^{n+2}},\ \text{MSE}=\frac{1}{9\times 16^{n+1}}.\tag{10}$$

The OAM [32] can produce a zero-mean error distribution, which is an appealing feature for applications with consecutive multiply-accumulate operations.

4.3 Hybrid Approximation

Several approximate designs combine the multipliers with different precisions together to adapt to varying accuracy requirements [92, 93]. These designs are considered as using hybrid approximation in this article. For example, accurate and approximate multipliers are combined to adjust the computational accuracy by selecting the appropriate multiplier in [92]. For the approximate multiplier, after detecting the number of consecutive 1's or 0's in the mantissa, the mantissa can then be rounded to 1 or 2, which converts the multiplication to a shift operation. If a higher precision is required, the accurate multiplier is then invoked to conduct the calculation. Reference [93] used the sum of two mantissas to approximate the multiplication. A tuning strategy is proposed to decide the working mode of the multiplier by detecting the number of consecutive bits in the inputs. However, such methods heavily rely on an accurate or high-precision multiplier, which significantly increases the circuit area.

5 APPROXIMATE MULTIPLIERS WITH ARCHITECTURE-LEVEL APPROXIMATION

As discussed in Section 2.2, a conventional multiplier typically involves three stages, *i.e.*, data input, partial product generation, and accumulation. In addition, an encoding stage is often deployed to further reduce the number of partial products, *e.g.*, Booth encoding [107]. For approximate multipliers based on such an architecture, approximations can be introduced into any of the four aforementioned stages.

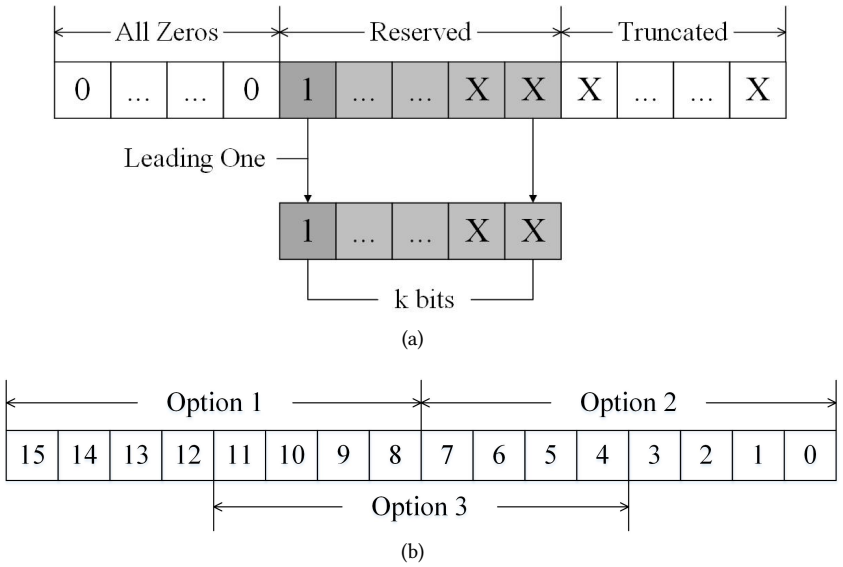

Fig. 12. (a) An example of DSM truncation; (b) An example of SSM truncation.

5.1 Approximation at Input

It is simple yet effective to introduce approximation in the input for approximate multipliers. For example, a few least significant bits (LSBs) can be removed to reduce the input bit-width, which has a lower impact on the result than removing the MSBs [36, 43–45]. In general, there are two types of input truncation schemes, the dynamic segment method (DSM) and static segment method

(SSM) [45]. DSM segments the data according to the leading one, while SSM is based on a given segmentation option. For example, in Fig. 12(a), DSM keeps k consecutive bits from the leading one of an unsigned number. The parameter k determines the level of accuracy loss for an approximate multiplier. In contrast, SSM in Fig. 12(b) provides a few pre-determined (*i.e.*, *static*) options when truncating the input. The options can be either leading k bits (option 1) or last k bits (option 2), as suggested in [45]. It is also possible to keep the bits in the middle (option 3) as a trade-off (Fig. 12(b)). Unlike DSM, SSM requires less hardware resources but may include more redundant bits. As shown in the Dynamic Range Unbiased Multiplier (DRUM) in [43], the additional support to DSM requires two extra Leading-One Detectors (LODs), two extra encoders, and one extra barrel shifter. Moreover, the last bit of the reserved part is always set to one for unbiasedness.

5.2 Approximation at Partial Product Generation

Kulkarni *et al.* proposed an under-designed multiplier (UDM) architecture, which brings approximation into the partial product generation stage [24]. UDM partitions both multiplier and multiplicand into 2 parts, and then implements inaccurate 2×2 multiplication blocks. As shown in Fig. 13, each partial product can be produced by an approximate multiplier.

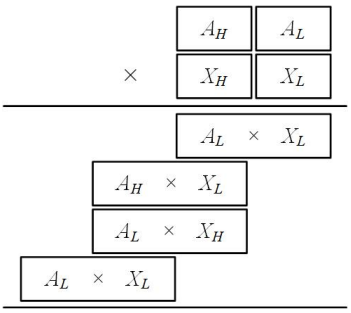

Fig. 13. An example of UDM in [24].

Another alternative to partial product generation is to introduce an intermediate variable to replace the partial products (*a.k.a.* altered partial product (APP)) and then conduct approximations [46–48]. As discussed in Section 2.2, a partial product can be generated using an AND gate:

$$pp_{m,n} = x_m \cdot y_n \ , \tag{11}$$

where x_m and y_n represent the m^{th} and the n^{th} bit of the two inputs x and y, respectively. Similar as carry look-ahead adder, the propagate and the generate signals can be defined as:

$$p_{m,n} = pp_{m,n} + pp_{n,m} \ , \tag{12}$$

$$g_{m,n} = pp_{m,n} \cdot pp_{n,m} \ . \tag{13}$$

Since the generate signals are possibly all 0's, they can be compressed column-wise using an OR gate. The propagate signals can be computed using approximate adders to achieve a more compact design than the original multiplier. Yang *et al.* employed a similar idea of using two signals of approximate sum and error recovery vector to approximate the partial product [47].

5.3 Approximation at Accumulation

In this section, we discuss the related works on approximation at the accumulation stage. First, we focus on the basic modules used at this stage, including approximate compressors and approximate

1-bit adders. Then, we review the works on the allocation of different approximate compressors in the entire accumulation stage.

5.3.1 Approximate Compressors. Instead of the exact compressors in Section 2.3, many approximate compressor designs have been proposed to improve the energy efficiency [46, 47, 49, 51–53, 83, 108–113]. We divide them into two categories, low-order and high-order approximate compressors. The low-order approximate compressors include approximate 1-bit half adder, approximate 1-bit full adder, and approximate 4-2 compressors. Taking the error rate into account, approximate 4-2 compressors with different approximation levels were designed to realize power efficient Dadda multipliers [53, 83]. One structure of the proposed approximate 4-2 compressors in [83] is shown in Fig. 14(a). Venkatachalam *et al.* took the error difference into account to design an approximate 1-bit half adder, an approximate 1-bit full adder, and an approximate 4-2 compressor. The structure of the approximate 4-2 compressor is shown in Fig. 14(b). In order to provide more flexibility for accuracy-power trade-off, Akbari *et al.* designed four approximate 4-2 compressors with different approximation levels, each of which can flexibly switch between the exact and the approximate operating modes [109]. A survey on some approximate 4-2 compressors is presented in [111] together with their performance comparison.

The second category of the approximate compressors is a high-order approximate compressor, which has at least 5 inputs. By modifying the truth table of the exact compressors, an approximate 5-2 compressor with significantly fewer transistors was proposed in [110]. Tung *et al.* proposed to design more general n-2 approximate compressors ($n \geq 5$) to further reduce the delay and power of the approximate multiplier [52]. Instead of using a 2-bit output, Marimuthu *et al.* [112] designed an approximate 15-4 compressor, where four different types of approximate 5-3 compressors were used as basic modules. Esposito *et al.* proposed more general approximate n-$\lceil n/2 \rceil$ compressors, where the outputs of these compressors are equally weighted [49]. Compared with the previously proposed unequally-weighted approximate compressors [46, 47, 51–53, 83, 108–112], these equally-weighted approximate compressors have smaller area, and their errors are easier to analyze.

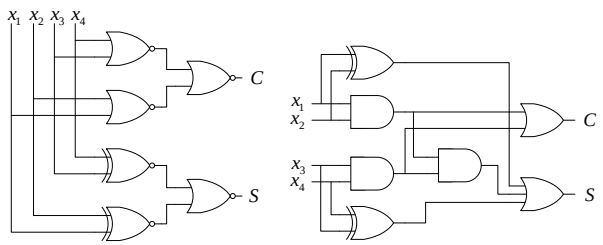

Fig. 14. Two approximate 4-2 compressors proposed by (a) Momeni *et al.* [83]; and (b) Venkatachalam *et al.* [46], where C is the carry signal and S is the sum signal.

However, all the prior approximate compressors were manually designed. To provide a general solution to approximate multiplier design and optimize over a number of hardware cost metrics, Wang *et al.* proposed a method called MinAC to automatically generate minimal-area approximate 4-2 compressors [113]. MinAC formulates the approximate circuit design as an exact synthesis problem [114], which introduces proper encoding variables and constraints to determine the connections of the gates, the gate functions, and the output gates of the circuit. For example, for any approximate 4-2 compressor in Fig. 14, its error metric such as the MED can be calculated

and denoted as e_b. Then, MinAC can be applied to synthesize an area-optimal approximate 4-2 compressor with an MED no larger than e_b. Fig. 15 shows the structures of the two area-optimal approximate 4-2 compressors obtained by MinAC. Obviously, compared to the original designs in Fig. 14, MinAC can synthesize approximate 4-2 compressors with much fewer gates and smaller area.

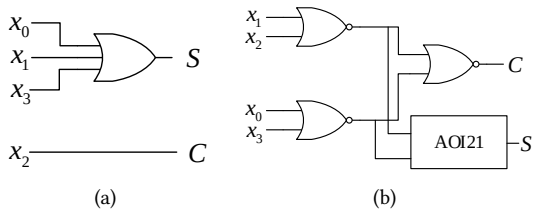

Fig. 15. The optimized structures by MinAC for the two approximate 4-2 compressors in Fig. 14: (a) Optimized design for the approximate compressor in [83]; and (b) Optimized design for the approximate compressor in [46].

5.3.2 Allocation of Approximate Compressors in the Compressor Tree. In the conventional multipliers, exact compressors, such as half adders, full adders, and 4-2 compressors, are applied to accumulate the partial product array. Researchers have proposed efficient allocation schemes to use as few compressors as possible during the accumulation stage in order to minimize area and delay. However, the traditional accumulation schemes are not applicable to approximate multipliers as they do not consider the accumulated errors in the tree structure. To address this issue, Venkatachalam *et al.* proposed a reduction scheme that uses approximate compressors to compress the partial product array to two rows, which are then added up by an RCA [46]. Jiang *et al.* further proposed to ignore the carry signals in adders to reduce the critical path delay [115]. However, the error may become non-negligible if completely relying on approximate compressors during accumulation. Thus, many researchers proposed to separate the partial product array column-wise to two or three groups, as shown in Fig. 16. Due to the difference in the significance of groups, each group can be assigned with different levels of approximation [47, 49–54]. While the first group is always allocated with exact compressors, the second group can be allocated with either approximate compressors using equally weighted outputs [49, 50] or high-order approximate compressors with error recovery [51, 52]. For the last group, one straightforward approach is to completely ignore it [47, 53]. To improve the accuracy, Tung *et al.* proposed to use OR gates for the last group [52]. To provide more flexibility, Mahdiani *et al.* divided the partial product array into four groups through the horizontal and vertical slicing in a broken-array multiplier (BAM), as shown in Fig. 17 [101]. The partial products on the right of vertical break level (VBL) or above the horizontal break level (HBL) are then ignored. In other words, only the partial products on the bottom left are used for calculation. Apparently, the approximation level can be adjusted by tuning VBL and HBL.

However, most prior works still allocate approximate compressors in an ad hoc manner or limited to some specific types of compressors. It is desirable to automate the allocation for different kinds of approximate compressors. For this purpose, Xiao *et al.* proposed a general framework for approximate compressor allocation called OPACT, which converts the allocation problem to an integer programming (IP) problem [95]. The proposed framework not only accounts for the trade-off between area and accuracy, but also optimizes the connection order of different compressors.

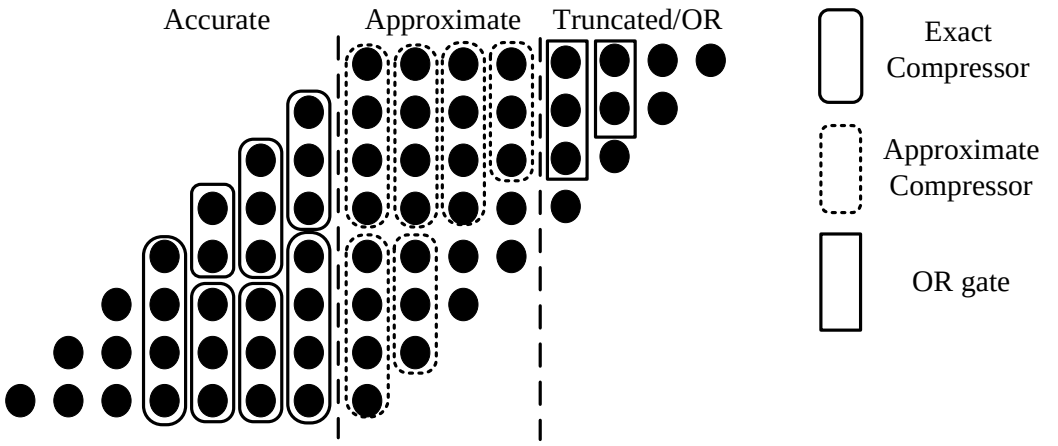

Fig. 16. An example of partial product array that is divided into three groups with different levels of approximation.

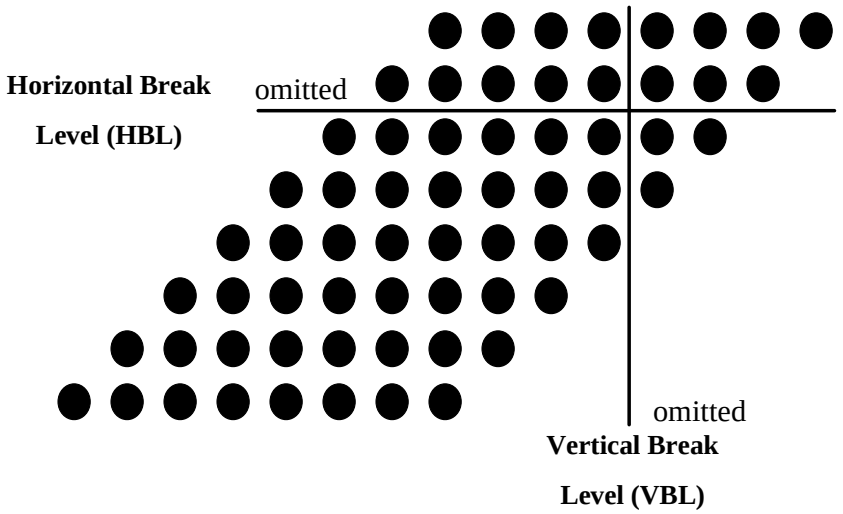

Fig. 17. An example of broken-array multiplier (BAM) [101].

5.4 Approximation at Booth Encoding

Booth encoding is often used to reduce the number of partial products, which can be generated in parallel at the cost of additional area. The radix-4 Booth algorithm is a common option deployed for high bit-width multipliers [116]. Qian *et al.* proposed an approximate Wallace-Booth multiplier with approximate modified Booth encoding (MBE), approximate 4-2 compressors, and approximate Wallace tree [89]. In addition to the radix-4 algorithm, the radix-8 Booth algorithm is also widely used to further reduce the number of partial products. However, the encoder in a radix-8 algorithm may generate odd values including 3 and -3 and hence needs additional adders to compute the odd multiples of multiplicands when generating the partial products. This step leads to an increased delay. To reduce the delay, Jiang *et al.* suggested an approximate adder to generate the odd multiples

for multiplication [56], which can reduce the delay of carry propagation as a trade-off between speed and accuracy.

6 APPROXIMATE MULTIPLIERS WITH CIRCUIT-LEVEL APPROXIMATION

This section reviews a few general circuit-level approximation techniques applicable to various architectures or algorithms, including Boolean rewriting, gate-level pruning, evolutionary circuit design, and voltage over-scaling (VOS).

6.1 Boolean Rewriting

Boolean rewriting modifies Boolean algebra expressions to simplify the circuit and is frequently utilized to approximate the basic modules in a multiplier. Karnaugh map (K-map) modification is a commonly used technique in this category. The basic idea of K-map is to group the adjacent cells with the same logic values as much as possible. However, it is quite common in practice that one or more cells cannot be grouped, causing additional logic and hence area. Thus, the approximation to K-map can be introduced by modifying the adjacent cells to the same value so that they can be grouped to obtain a more compact representation. For example, the approximate multiplier UDM discussed in Section 5.2 is comprised of a 2×2 multiplication module [24], which can be designed through K-map modification. By modifying the K-map as in Fig. 18, the basic block can act as both a partial product generator and a compressor with an error rate of 1/16 [24]. As shown in Fig. 19, when compared to the accurate logic implementation, the approximate implementation needs much fewer logic gates (37.5% reduction) with a shorter critical path.

A_1A_0 \ B_1B_0	00	01	11	10
00	000	000	000	000
01	000	001	011	010
11	000	011	111	110
10	000	010	110	100

Fig. 18. An example of modifying K-map to achieve more compact design [24]: the accurate output '1001' at the cell $(A_1A_0B_1B_0) = (1111)$ is changed to '111'.

The K-map modification can also be applied to other basic modules, such as adders [57], compressors [51, 54], and Booth encoding modules [55, 58, 89]. For example, Yin *et al.* used K-map modification to design an approximate modified Booth encoding (AMBE) module [58]. With the modified K-map in Fig. 20, the original expression for the modified Booth encoding algorithm:

$$PP_j = (X_{2i} \oplus X_{2i-1})(X_{2i+1} \oplus Y_j) + \overline{(X_{2i} \oplus X_{2i-1})}(X_{2i+1} \oplus X_i)(X_{2i+1} \oplus Y_{j-1}) \ , \tag{14}$$

can be simplified to [58]:

$$PP_j^{'} = (X_{2i} \oplus X_{2i-1})(X_{2i+1} \oplus Y_j) \ . \tag{15}$$

In addition to K-map modification, a Boolean expression can also be directly changed to create a more compact expression, such as the approximate adders SOA in [105] and lower-part-or adder (LOA) in [101], where the former sets the least significant part of the addition to one and the latter computes with OR gates.

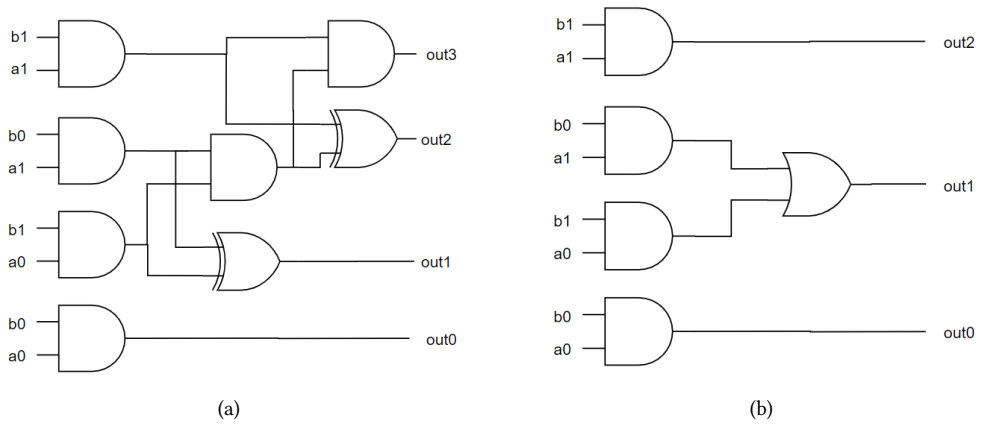

Fig. 19. Comparison on the implementations of the 2×2 multiplier module: (a) Accurate logic implementation; (b) Approximate logic implementation [24].

Y_jY_{j-1} \ $X_{2i+1}X_{2i}X_{2i-1}$	000	001	011	010	110	111	101	100
00	0	0	0	0	1	1	1	0
01	0	0	0	0	1	1	1	0
11	0	1	1	1	0	0	0	0
10	0	1	1	1	0	0	0	0

Fig. 20. K-map modification of AMBE [58]: The circled values in the table are flipped to enable a more compact representation.

6.2 Gate-Level Pruning

Gate-level pruning provides an alternative to simplify the netlist. It is based on the probabilistic pruning, which prunes less active gates from a circuit with a limited accuracy loss [90]. Schlachter *et al.* proposed to transform the circuit to a graph and prune the nodes with the lowest significance-activity product (SAP) during synthesis [91]. The term "significance" indicates the importance of each node/gate, while "activity" refers to the toggling rate of the gate. The significance for the output nodes is user-defined and then backward propagated to calculate the significance of other gates. The activity of a node can be extracted from the switching activity interchange format (SAIF) file, which presents the toggle counts of wires. The digital design flow with gate-level pruning is shown in Fig. 21.

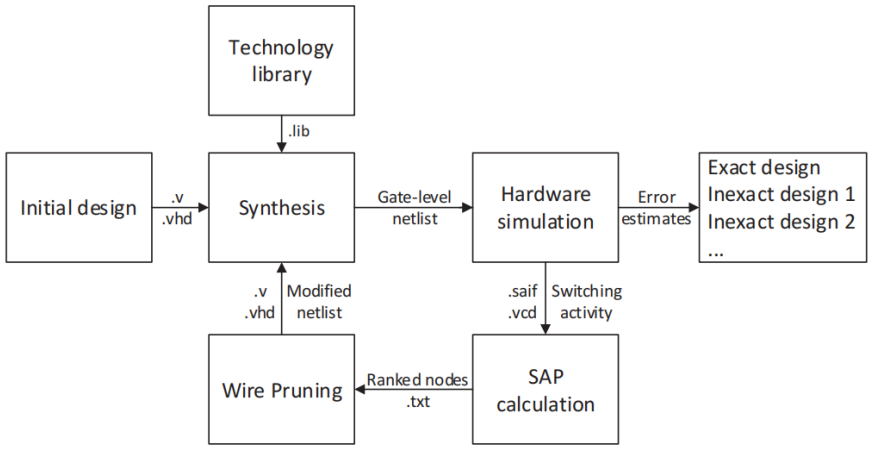

Fig. 21. An example of gate-level pruning in a digital design flow [91].

6.3 Evolutionary Circuit Design

Evolutionary algorithms, especially genetic programming, have been utilized to design digital circuits including approximate designs [96, 97, 117, 118]. Cartesian Genetic Programming (CGP) that uses graph representations is a flexible form of the genetic programming [119]. Based on CGP, circuits are represented by node arrays, where a node represents a basic logic function such as AND and OR. The design process starts with the circuit of an existing conventional multiplier. Then, a group of circuits are randomly generated from the initial one by applying some operations analogy to the naturally occurring genetic operations. Each circuit is evaluated with multiple objectives including error, delay, and power. Based on the evaluation, satisfactory circuits are selected as a new generation of circuits. When the maximum number of generations is reached or the requirements of approximate multipliers are satisfied, the iteration is terminated and the connected nodes determine the multiplication implementation. Hrbacek *et al.* followed the scheme and applied Non-dominated Sorting Genetic Algorithm II (NSGA-II) to sort out approximate multipliers among the Pareto front [97]. Based on the previous work [97], reference [96] constructed a library containing 471 8-bit approximate multipliers.

6.4 Voltage Over-scaling

Voltage scaling is another commonly used method to reduce the power consumption [29]. In general, the supply voltage needs to be higher than $V_{dd-crit}$, which is the minimum supply voltage to ensure the correct timing of the critical path [30, 99, 102, 120]. While voltage over-scaling (VOS) effectively reduces the power, timing violation induced errors is inevitably introduced [99]. Thus, Lau *et al.* provided different energy budgets for each column of the array multiplier in order to minimize the computation error [102]. Since VOS mainly impacts the critical and near-critical paths, it is desired to adjust the architecture of each computation module to achieve a shorter critical path and mitigate the impact of low supply voltage [30, 99]. Liu *et al.* also proposed an analytical model to assess the VOS-induced computation errors, which can then be used to select the corresponding architecture and setup [30].

7 EVALUATION

In this section, we evaluate the representative approximate multipliers presented in the previous sections in terms of accuracy and circuit characteristics (including delay, power, and area). Then, these approximate multipliers are applied in two machine learning applications by replacing the original exact multipliers with them. The classification accuracy and energy consumption of the representative designs are presented for comparison.

7.1 Quantitative Comparison Among Multipliers

Given the above discussion on various techniques introduced for approximate multiplier design, it is necessary to further quantitatively compare their performance under the same condition. Thus, the following subsections compare the designs under three categories, *i.e*. fixed-point unsigned, fixed-point signed, and floating-point multipliers, where 16-bit is selected for fixed-point. It is noted that all the designs are evaluated with the same setup in the following experiments. In particular, we use mean relative error distance (MRED) and normalized mean error distance (NMED) as accuracy metrics. The circuit measurements include delay, power, and area. The approximate multipliers are implemented by Verilog and then synthesized by Synopsys Design Compiler [121] using the UMC40 library [122] under either delay-optimized or area-optimized mode. For each multiplier, we randomly generate 10-million input pairs following a uniform distribution and then run simulations in Verilator [123].

7.1.1 Fixed-Point Unsigned Multipliers. We have implemented the aforementioned representative fixed-point unsigned multipliers listed in Table 1. SSM [45] and DRUM [43] are implemented with the segment width 8. APP is implemented by altering partial products and globally applying the approximate reduction technique, while APP2 separates the partial product array in a column-wise manner into three groups, *i.e*., accurate, approximate, and truncated parts, with 7, 16, and 8 columns, respectively [46]. AW adopts the same partition scheme as APP2 for comparison, but uses an AND gate-based partial product generation scheme [51]. The approximate multiplier BAM is denoted as BAM(m, n), where m and n denote the numbers of omitted rows and columns when carrying out the accumulation [101]. OPACT [95], UDM [24], LM [11], Iterative LM (IterLM) [11] and Improved LM (ImprLM) [40] are implemented following the original works. Finally, we also implement the approximate logarithm-based multiplier (ALM) and its iterative variant, IALM, both using approximate adders in the design. The implementation of ALM uses the approximate adder SOA [105] that sets the 11 LSBs approximate. We denote it as ALM_SOA. The implementation of IALM uses three approximate adders, including two SOAs and one LOA [101] configured by setting the numbers of approximate bits as 5, 11, and 16, respectively. We denote it as IALM_SL.

Figs. 22–24 show the accuracy and the circuit characteristics of multiple unsigned approximate multipliers when compared to an exact 16-bit multiplier IP obtained from DesignWare [124]. In Fig. 22, we show the errors of the approximate multipliers, measured by MRED and NMED. Thus, a lower bar indicates a higher accuracy. Figs. 23 and 24 show the delay/power/area reduction ratio over the exact design under the delay-optimized and the area-optimized modes, respectively, where the corresponding exact design is synthesized under the same mode. A higher positive bar indicates a larger delay, power, or area improvement. From Fig. 23, we can observe that under the delay-optimized mode, SSM shows better circuit characteristics than DRUM with comparable accuracy. Due to the approximation in the MSBs, APP has an MRED over 13% and an NMED about 5%, while its variant APP2 shows much improved accuracy with very similar circuit characteristics. Comparing APP2 and AW, the altered partial products obviously help the efficiency of the reduction procedure but at the cost of area and power. Among all the BAM multipliers, BAM(8,16) provides the best trade-off between accuracy and circuit cost. OPACT, configured with automatically optimized

compressor allocation and order connection, provides negligible error with moderate hardware savings. As for logarithm-based multipliers, the iterative schemes including IterLM and IALM_SL consume too much hardware cost for error compensation to achieve the desired accuracy. ALM_SOA improves the circuit characteristics with the approximate adder at the cost of moderate accuracy degradation.

As shown in Fig. 24, under the area-optimized mode, the results for most designs, *e.g.*, DRUM, APP, APP2, UDM, OPACT, LM, ALM_SOA, show an improvement in the area reduction ratio at the cost of a decreased delay reduction ratio, when compared to the delay-optimized cases. In contrast, the multipliers BAMs, AW, IterLM, and IALM_SL show improvements in delay reduction at the cost of smaller area and power reduction ratios. The reason for the decrease in the area reduction ratio lies in the fact that the reduction ratio is measured against the exact adder synthesized under the area-optimized mode. In that case, the area of the exact adder also reduces compared to the exact adder synthesized under the delay-optimized mode. **Such a comparison interestingly explores the sensitivity of the various approximation techniques to the synthesis mode**, which has not been well illustrated in the prior work.

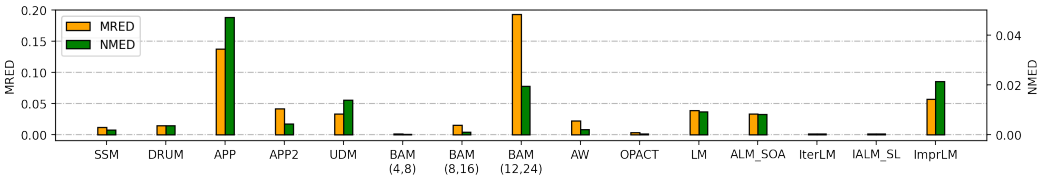

Fig. 22. Error comparison of approximate fixed-point unsigned multipliers.

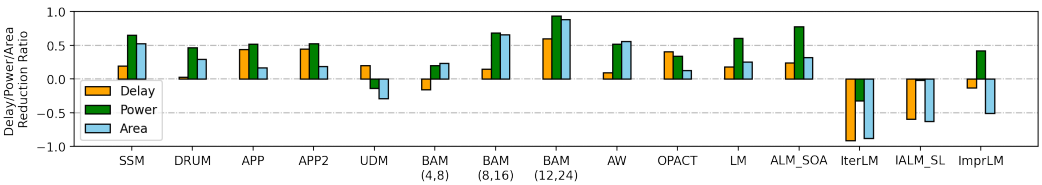

Fig. 23. Circuit characteristics comparison of delay-optimized approximate multipliers.

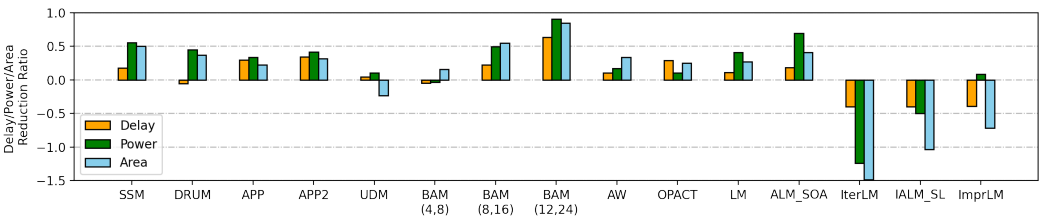

Fig. 24. Circuit characteristics comparison of area-optimized approximate multipliers.

7.1.2 Fixed-Point Signed Multipliers. For fixed-point signed approximate multiplication, Booth multipliers are widely used by adding an extra stage of Booth encoding to the conventional binary multiplier. Thus, the aforementioned architecture-level approximation techniques are applicable to Booth multipliers as well [17, 56, 98, 100, 103, 104]. Given the comprehensive comparison of the above techniques in Section 7.1.1, here we only focus on the comparison of the approximate techniques in Booth encoding (see Section 5.4). In particular, we choose R4SA, and its variants R4ABE1 and R4ABE2, which neglect the signal neg_cin in the last partial product line in comparison to the exact Booth radix-4 multiplier [55]. Here, the signal neg_cin is the additional logic value 1 added to the LSB when handling a negative number in 2's complement. R4ABE1 and R4ABE2 utilize different approximate encoders designed by K-map modification and are configured by setting the number of approximate LSBs as 15. The prefix of "R4" refers to the radix-4 approximate multiplier, which is compared to the exact radix-4 multiplier. Similarly, R8ATM is a radix-8 approximate multiplier by introducing an approximate re-coding adder for the computation of the expression $(Y + 2Y)$ and hence, it is compared to the exact radix-8 multiplier [56].

As we can see in Figs. 25–27, R4SA has a small error but distinct improvement in circuit characteristics. The ignorance of neg_cin in the last partial product line reduces the extra hardware cost for accumulation. Since R4ABE1 and R4ABE2 introduce additional approximation in the encoders based on R4SA, we mainly focus on the differences among them. Both R4ABE1 and R4ABE2 have a larger MRED and NMED than R4SA, but they produce mixed results for different error measures (i.e., MRED and NMED) when compared to each other. When the delay is optimized, R4ABE2 presents a higher improvement in delay and R4ABE1 is superior in power and area. When the area is optimized, R4ABE1 and R4ABE2 demonstrate improvement in all the three circuit metrics. On the other hand, R8ATM shows a negligible delay improvement over the exact radix-8 multiplier under the delay-optimized mode. Under the area-optimized mode, R8ATM is found to have more balanced improvements in circuit characteristics.

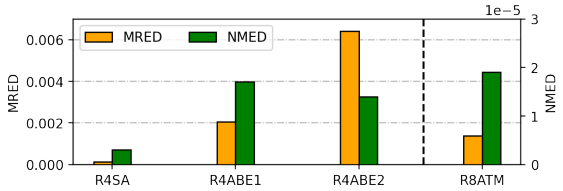

Fig. 25. Error comparison of approximate fixed-point signed multipliers.

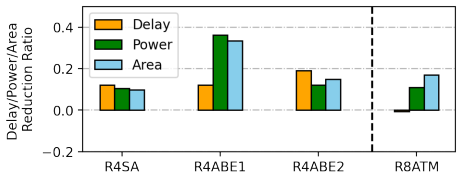

Fig. 26. Circuit characteristics comparison of delay-optimized approximate fixed-point signed multipliers.

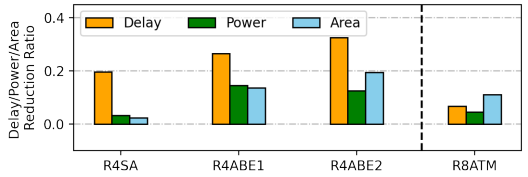

Fig. 27. Circuit characteristics comparison of area-optimized approximate fixed-point signed multipliers.

7.1.3 Floating-Point Multipliers. Many floating-point multipliers introduce approximation at the algorithm level, *e.g.*, through linearization or hybrid approximation. We implement and compare the representative approximation techniques in this section, *i.e.*, ApproxLP [31], OAM [32], and RMAC [93]. An exact 32-bit floating-point multiplier IP from DesignWare [124] is used as the baseline. We denote the multipliers ApproxLP and OAM with the approximation level n as ApproxLP(n) and OAM(n), respectively. A smaller n indicates a higher approximation.

As shown in Figs. 28–30, when the approximation level decreases, both ApproxLP and OAM show a larger error with smaller hardware costs. In general, at the same approximation level, OAM has a smaller error, area, and delay than ApproxLP, while ApproxLP has a lower power dissipation. In Fig. 29, unlike ApproxLP showing stable improvements in circuit characteristics with an increased approximation level, OAM shows sharper changes in circuit characteristics from level-(2, 3, 4) to level-(0, 1). When considering area-optimized synthesis in Fig. 30, the circuit metrics show steady improvement with the increased approximation level for both ApproxLP and OAM. However, ApproxLP shows a negative delay improvement, simply indicating a longer delay than the exact multiplier design. RMAC maps the mantissa multiplication to the addition between the two inputs and shows a similar performance as ApproxLP(0) with a larger delay reduction ratio.

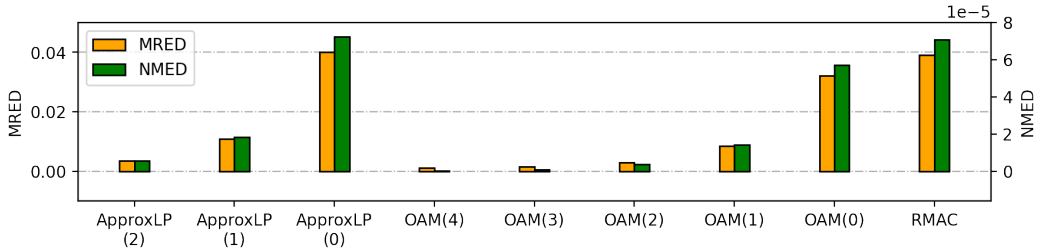

Fig. 28. Error comparison of approximate floating-point multipliers.

7.2 Application-Level Comparison

In addition to the generic evaluations, we further study the performance of the approximate multipliers in machine learning applications, which are typically considered multiplication-intensive but error-tolerant by its nature. Two representative machine learning applications are selected for comparison: one is a multi-layer perceptron (MLP) network with one hidden layer on the MNIST dataset, which is denoted as MNIST (MLP); the other is the convolutional neural network AlexNet on the CIFAR10 dataset, which is denoted as CIFAR10 (AlexNet). Both neural networks are pre-trained with accurate floating-point training infrastructure to obtain the trained weights in double precision. The baselines have been implemented in Verilog using the exact multipliers to find out the baseline classification accuracy, power, and delay of executing the two tasks, which can

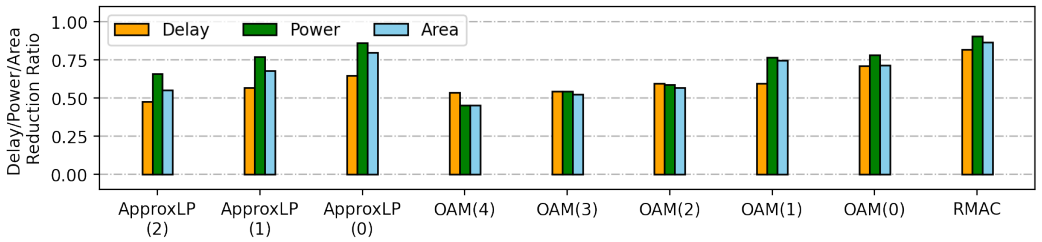

Fig. 29. Circuit characteristics comparison of delay-optimized approximate floating-point multipliers.

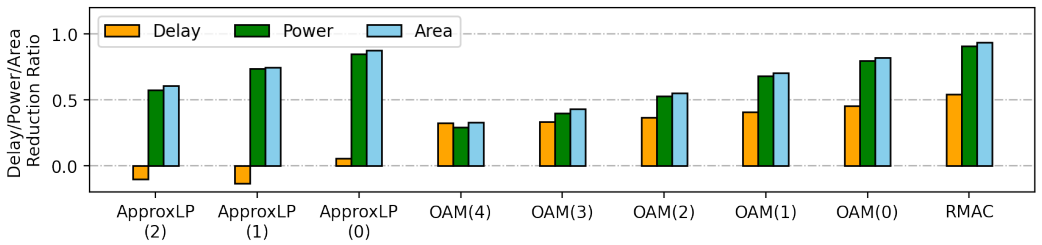

Fig. 30. Circuit characteristics comparison of area-optimized approximate floating-point multipliers.

be measured by Prime-Time PX. During the evaluation, the original exact multipliers are replaced by the approximate multipliers to evaluate their impact on both classification accuracy and energy efficiency. The accuracy loss is defined as the classification accuracy change due to the use of the approximate multipliers, while the energy efficiency is given by the power-delay-product (PDP). For the evaluations using fixed-point multipliers, post-training quantization has been applied to obtain the fixed-point weights [125]. For example, the weights and inputs are mapped to the range of $[0, 65535]$ for 16-bit fixed-point unsigned multipliers, and $[-32767, 32767]$ for 16-bit fixed-point signed multipliers.

7.2.1 Fixed-Point Unsigned Multipliers. This section studies the performance of various approximate fixed-point unsigned multipliers. Fig. 31 presents the power reduction ratio when deploying the approximate multiplier in MNIST (MLP) and CIFAR10 (AlexNet) in comparison to the cases of using the exact multiplier. Other than a few logarithm-based designs requiring extra power consumption for error compensation, most designs can achieve quite large power reductions with the largest power saving above 95%. Thus, the application-level accuracy need to be further considered to determine the most suitable approximate multipliers for the machine learning applications. Fig. 32 shows the classification accuracy loss versus the energy efficiency trade-off for various approximate multipliers on the two applications. The PDP reduction ratio is defined as the relative PDP change over the original PDP, where a positive PDP reduction ratio indicates improved energy efficiency and a negative ratio indicates reduced energy efficiency compared to the original design. On the other hand, the higher accuracy loss indicates a worse classification performance. Thus, the designs that are closer to the lower right corner are preferred. It can be observed that, while OPACT, BAM(8,16), SSM, ALM_SOA, AW, APP2, BAM(4,8), DRUM, and LM can achieve good improvements in energy efficiency with a limited or almost negligible accuracy loss on the MNIST (MLP), only OPCAT, DRUM, and BAM(4,8) can keep good accuracy with large improvements in energy efficiency on the CIFRA10 (AlexNet). This simply indicates that the potential approximation

design space actually changes for different algorithms and demands more in-depth understanding of the approximation error propagation when executing the algorithm. As discussed earlier, iterative log-based designs do not perform well in terms of energy efficiency in both cases, which are located close to the lower left corner of the plots.

Moreover, if we compare Fig. 22 and Fig. 32(a), it is found that, in general, the multipliers with smaller MREDs tend to cause smaller accuracy losses on MNIST (MLP). Both APP and BAM(12,24) are located on the upper right corner of Fig. 32(a) due to their large MREDs in the generic evaluation. However, when the application (or algorithm) becomes more complex, as shown in Fig. 32(b), more designs even with moderate MREDs in Fig. 22 show very poor performances in terms of accuracy on CIFAR10 (AlexNet). The difference is partly due to the shallowness of MLP network, where the approximation errors can hardly get accumulated. In addition, some multipliers, like ALM_SOA and LM, introduce one-sided errors that can become more difficult to be cancelled out during the approximation error propagation within the AlexNet. Finally, the multipliers such as BAM(8,16), AW, and APP2 tend to introduce larger relative errors for small inputs with their aggressive approximations in the least-significant bits. **Thus, when deploying an approximate multiplier, instead of simply selecting the one based on its MRED, we should choose one with balance among error bias, approximation aggressiveness, and its fitness to a particular algorithm.**

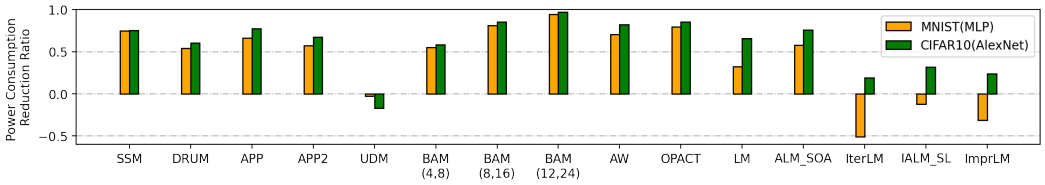

Fig. 31. Power reduction of deploying approximate fixed-point unsigned multipliers in MNIST (MLP) and CIFAR10 (AlexNet) when compared to the case of using the exact multiplier.

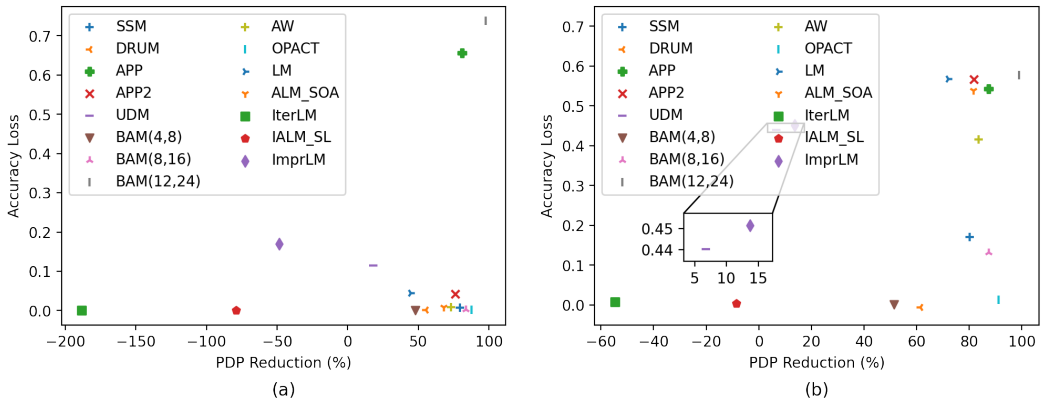

Fig. 32. Classification accuracy loss versus PDP reduction ratio for the delay-optimized approximate 16×16 unsigned multipliers on (a) MNIST (MLP) and (b) CIFAR10 (AlexNet).

7.2.2 Fixed-Point Signed Multipliers. Fig. 33 presents the comparison of power reduction among different approximate fixed-point signed multipliers, with the baseline design using the exact signed multipliers. Similar as in Fig. 26, all the designs show power reductions over the exact multipliers, where R4ABE1 can achieve the most power saving. Fig. 34 compares the trade-off between classification accuracy loss and PDP reduction ratio for the two applications using different approximate signed fixed-point multipliers. When comprehensively considering the accuracy loss and PDP, R4ABE1 and R4ABE2 are superior to the other multipliers. R8ATM has the worst performance in terms of PDP reduction due to its more complicated encoding.

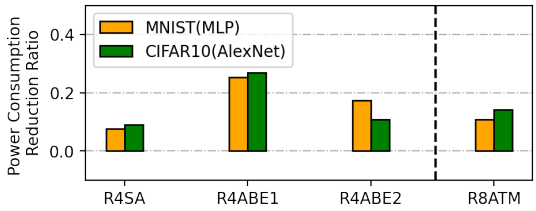

Fig. 33. Power reduction of deploying approximate fixed-point signed multipliers in MNIST (MLP) and CIFAR10 (AlexNet) when compared to the case of using the exact multiplier.

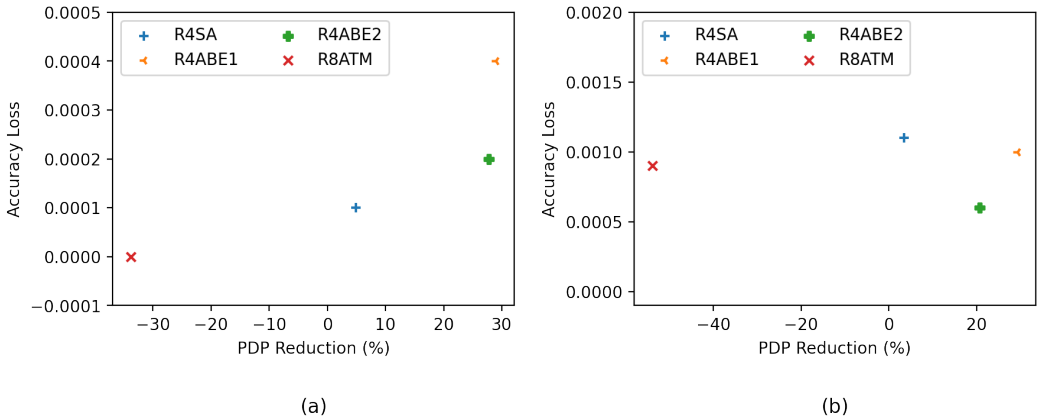

Fig. 34. Classification accuracy loss versus PDP reduction ratio for the delay-optimized approximate 16 × 16 signed multipliers on (a) MNIST (MLP) and (b) CIFAR10 (AlexNet).

7.2.3 Floating-point Multipliers. Finally, we investigate the results of designs using approximate floating-point multipliers. As shown in Fig. 35, all the approximate floating-point multipliers can achieve dramatic power reduction. Such power saving eventually contributes to the large PDP reduction in Fig. 36, in which the smallest energy efficiency improvement is more than 65% for OAM(4) on MNIST (MLP). Due to the error tolerance of the two applications and the large dynamic range of the floating-point multipliers, all the applications adopting the multipliers in this category have negligible accuracy losses, where the worst case is CIFAR10 (AlexNet) using RMAC with an accuracy loss of only 0.024. Moreover, the impact of approximation level is very limited when

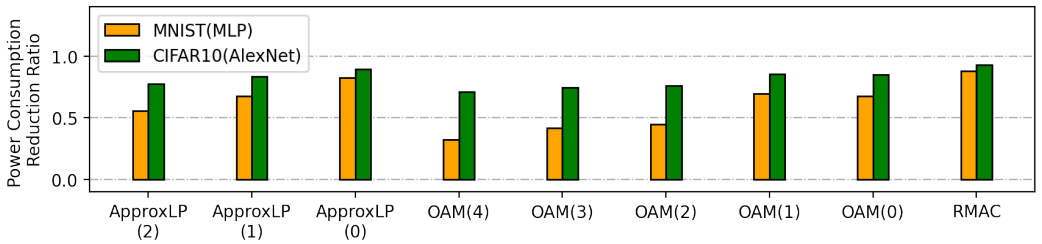

Fig. 35. Power reduction of deploying approximate floating-point multipliers in MNIST (MLP) and CIFAR10 (AlexNet) when compared to the case of using the exact multiplier.

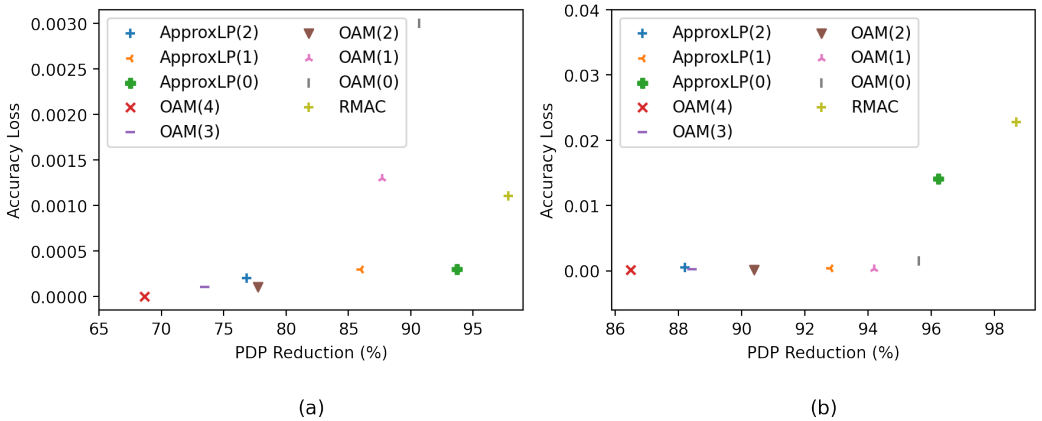

Fig. 36. Classification accuracy loss versus PDP reduction ratio for the delay-optimized approximate floating-point multipliers on (a) MNIST (MLP) and (b) CIFAR10 (AlexNet).

comparing OAM or ApproxLP with different approximation levels. Thus, compared to the fixed-point counterparts, **the larger dynamic range in approximate floating-point multipliers actually contributes to a better trade-off of accuracy and energy efficiency on the two machine learning applications.**

8 DISCUSSIONS

Both the reviews and evaluations in the previous sections demonstrate the growing popularity and maturity of the research on approximate multipliers. However, there are still various challenges to be overcome, ranging from the design methodology, tools, to applications. This section discusses a few potential research directions that may shed light on easier and more efficient integration of approximate multipliers in the digital design flow in the future.

- **Top-down design methodology**: The demand for approximate multipliers is application driven to achieve higher energy efficiency at the cost of accuracy. On the other hand, as shown in the last section, the accuracy of approximate multipliers is not equivalent to the accuracy at the application level. For example, the metrics like MRED and NMED measuring the error distance between approximate and exact results do not correlate well with the application-level quality measures, such as the peak signal-to-noise ratio (PSNR) for image processing and the average precision (AP) for object detection. Such a mismatch inevitably

incurs additional design effort or over-design when deploying approximate multipliers. Thus, for a top-down design methodology, it is necessary to build a link between the application-level specification and error metrics of approximate multipliers. In addition, when both exact and approximate arithmetic circuits are used, the data flow needs to be revised to support such mixed precision [126]. Both processes demand a general design methodology to support the top-down design flow.

- **Design automation methods**: Most existing approximate multipliers were designed manually. However, with the various approximation options at different design levels, the design space for approximate multipliers is huge. Thus, the design of an optimized approximate multiplier calls for automation methods that systematically explore the design space. In this direction, some methods have been proposed for the architecture-level and circuit-level approximation, such as the automatic synthesis of approximate 4-2 compressors [113] and optimized allocation of approximate compressors in the compression stage [95]. These automatic synthesis methods are able to obtain better designs than the existing manual designs. However, given the large and complicated design space of approximate multipliers, there is still room for the development of more efficient and powerful design automation methods.
- **Algorithm-circuit co-optimization**: Finally, many applications such as machine learning are error-tolerant. They can then utilize approximate multipliers to achieve the energy efficiency improvements without much accuracy loss. However, as shown in the last section, there are still many unknowns on the interplay between the algorithm and the underlying approximate multiplier. In particular, it is highly desired to conduct research on designing approximation-friendly algorithms to fully utilize the benefits of the approximate multipliers. Such a study calls for an understanding of the approximation theory, error propagation, and error representation in the algorithms to define the appropriate design space. It is clear that the design space highly depends on the underlying approximate multiplier circuit. Thus, more research efforts need to be placed upon the interplay between the approximate algorithm design space and the characteristics of the underlying approximate multiplier circuit.

9 CONCLUSIONS

Widespread use of multiplication in error-tolerant applications enables significant improvements of energy efficiency when using approximate multipliers. In this article, we review approximate multipliers at the algorithm, architecture, and circuit levels. Detailed experimental results are presented on both generic and machine learning applications to help understand the advantages and disadvantages of various techniques at different design levels.

At the algorithm level, logarithm-based, linearization-based, and hybrid approximations are discussed. Mitchell's logarithmic multiplier shows significant hardware savings but with one-sided errors. The iterative logarithmic ones reduce the error, but suffer from hardware overhead for error compensation. The linearization scheme saves substantial energy and area with negligible errors, and also provides an error distribution with zero mean. The hybrid scheme collaboratively utilize highly approximated multipliers and more accurate ones, which may increase the circuit area. Although the reviewed works of the first two techniques focus on fixed-point and floating-point multiplications, respectively, there are some similarities between them. Both the logarithm-based and the linearization-based algorithms transform the multiplication to simpler linear computations, despite their different mechanisms. The logarithm-based method exploits the fact that in the logarithmic domain, a multiplication becomes a linear addition, while the linearization-based method directly performs piecewise-linear approximation for the non-linear multiplication. With the same need for locating the leading one, both of them fit floating-point multipliers well due to

the normalized mantissa, but it may take extra hardware costs for their utilization in fixed-point multipliers.

At the architecture level, various approximation strategies are presented and discussed at different stages of a conventional exact multiplier. Truncating input operands or partial products is a simple yet effective way to reduce the hardware cost. However, static truncation may cause a large relative error for small operands, which can affect the solution quality of complex applications. Applying the altered partial products enables the acceleration of accumulation, while the altering process increases the circuit area to some extent. Using approximate compressors in the accumulation stage is another effective way to design approximate multipliers. However, the allocation of approximate compressors in the accumulation stage and the determination of their connection orders are complicated problems that deserve further study.

At the circuit level, approximation techniques can be applied with the aforementioned approximation methods at the architecture and the algorithm levels. Boolean rewriting is often utilized to simplify the circuit of basic modules adopted in approximated multipliers. Gate-level pruning and evolutionary circuit design are both objective-oriented, and thus the trade-off between hardware and accuracy can be adjusted to meet different requirements. Voltage over-scaling can reduce energy consumption effectively, while the induced errors due to timing violations are often non-negligible, since the timing violations occur on the critical paths, which typically involve computation over the MSBs.

Furthermore, the generic and application-level evaluations give two insights: (1) various approximation techniques show different sensitivity to the synthesis mode, and thus may present circuit improvements of different degrees under different synthesis modes; (2) the selection of approximate multipliers for specific applications is concerned with not only the basic error metrics such as NMED and MRED, but also error bias, approximation aggressiveness, and its fitness to the algorithm.

REFERENCES

[1] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[2] Honglan Jiang, Francisco Javier Hernandez Santiago, Hai Mo, Leibo Liu, and Jie Han. Approximate arithmetic circuits: A survey, characterization, and recent applications. *Proceedings of the IEEE*, 108(12):2108–2135, 2020.

[3] Vaibhav Gupta, Debabrata Mohapatra, Sang Phill Park, Anand Raghunathan, and Kaushik Roy. IMPACT: IMPrecise adders for low-power approximate computing. In *IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 409–414. IEEE, 2011.

[4] Jie Han and Michael Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *2013 18th IEEE European Test Symposium (ETS)*, pages 1–6, 2013.

[5] Mohsen Imani, Abbas Rahimi, and Tajana S. Rosing. Resistive configurable associative memory for approximate computing. In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, DATE '16, page 1327–1332, San Jose, CA, USA, 2016. EDA Consortium.

[6] Swagath Venkataramani, Vinay K. Chippa, Srimat T. Chakradhar, Kaushik Roy, and Anand Raghunathan. Quality Programmable Vector Processors for Approximate Computing. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, page 1–12, New York, NY, USA, 2013. Association for Computing Machinery.

[7] Weiqiang Liu, Fabrizio Lombardi, and Michael Shulte. A retrospective and prospective view of approximate computing [point of view. *Proceedings of the IEEE*, 108(3):394–399, 2020.

[8] Jianing Deng, Zhiguo Shi, and Cheng Zhuo. Energy-efficient real-time UAV object detection on embedded platforms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):3123–3127, 2019.

[9] Gokul Govindu, Ling Zhuo, Seonil Choi, and Viktor Prasanna. Analysis of high-performance floating-point arithmetic on FPGAs. In *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, page 149, 2004.

[10] Robert K Yu and Gregory B Zyner. 167 MHz radix-4 floating point multiplier. In *Proceedings of the 12th Symposium on Computer Arithmetic*, pages 149–154, 1995.

[11] John N Mitchell. Computer multiplication and division using binary logarithms. *IRE Transactions on Electronic Computers*, EC-11(4):512–517, 1962.
[12] YC Lim. Single-precision multiplier with reduced circuit complexity for signal processing applications. *IEEE transactions on Computers*, 41(10):1333–1336, 1992.
[13] Michael J Schulte and Earl E Swartzlander. Truncated multiplication with correction constant [for DSP]. In *Proceedings of IEEE Workshop on VLSI Signal Processing*, pages 388–396. IEEE, 1993.
[14] Jer Min Jou, Shiann Rong Kuang, and Ren Der Chen. Design of low-error fixed-width multipliers for DSP applications. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 46(6):836–842, 1999.
[15] Lan-Da Van, Shuenn-Shyang Wang, and Wu-Shiung Feng. Design of the lower error fixed-width multiplier and its application. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 47(10):1112–1118, 2000.
[16] Shyh-Jye Jon and Hui-Hsuan Wang. Fixed-width multiplier for DSP application. In *Proceedings 2000 International Conference on Computer Design*, pages 318–322. IEEE, 2000.
[17] Kyung-Ju Cho, Kwang-Chul Lee, Jin-Gyun Chung, and Keshab K Parhi. Design of low-error fixed-width modified booth multiplier. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(5):522–531, 2004.
[18] L-D Van and Chih-Chyau Yang. Generalized low-error area-efficient fixed-width multipliers. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 52(8):1608–1619, 2005.
[19] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Low precision arithmetic for deep learning. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015.
[20] Ku He, Andreas Gerstlauer, and Michael Orshansky. Circuit-Level Timing-Error Acceptance for Design of Energy-Efficient DCT/IDCT-Based Systems. *Circuits and Systems for Video Technology, IEEE Transactions on*, 23:961–974, 06 2013.
[21] Mohsen Imani, Yeseong Kim, Abbas Rahimi, and Tajana Rosing. ACAM: Approximate Computing Based on Adaptive Associative Memory with Online Learning. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ISLPED '16, page 162–167, New York, NY, USA, 2016. Association for Computing Machinery.
[22] Mohsen Imani, Shruti Patil, and Tajana S. Rosing. MASC: Ultra-Low Energy Multiple-Access Single-Charge TCAM for Approximate Computing. In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, DATE '16, page 373–378, San Jose, CA, USA, 2016. EDA Consortium.
[23] Mohsen Imani, Mohammad Samragh, Yeseong Kim, Saransh Gupta, Farinaz Koushanfar, and Tajana Rosing. RAPIDNN: In-Memory Deep Neural Network Acceleration Framework. *CoRR*, abs/1806.05794, 2018.
[24] Parag Kulkarni, Puneet Gupta, and Milos Ercegovac. Trading accuracy for power with an underdesigned multiplier architecture. In *2011 24th Internatioal Conference on VLSI Design*, pages 346–351. IEEE, 2011.
[25] Muhammad Shafique, Rehan Hafiz, Semeen Rehman, Walaa El-Harouni, and Jörg Henkel. Invited: Cross-layer approximate computing: From logic to architectures. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2016.
[26] Alexander Suhre, Furkan Keskin, Tulin Ersahin, Rengul Cetin-Atalay, Rashid Ansari, and A Enis Cetin. A multiplication-free framework for signal processing and applications in biomedical image analysis. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1123–1127, 2013.
[27] Shaghayegh Vahdat, Mehdi Kamal, Ali Afzali-Kusha, Massoud Pedram, and Zainalabedin Navabi. TruncApp: A truncation-based approximate divider for energy efficient DSP applications. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 1635–1638, 2017.
[28] Cheng Zhuo, Kassan Unda, Yiyu Shi, and Wei-Kai Shih. From Layout to System: Early Stage Power Delivery and Architecture Co-Exploration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, PP:1–1, 05 2018.
[29] Anantha P Chandrakasan and Robert W Brodersen. Minimizing power consumption in digital CMOS circuits. *Proceedings of the IEEE*, 83(4):498–523, 1995.
[30] Yang Liu, Tong Zhang, and Keshab K Parhi. Computation error analysis in digital signal processing systems with overscaled supply voltage. *IEEE transactions on very large scale integration (VLSI) systems*, 18(4):517–526, 2009.
[31] Mohsen Imani, Alice Sokolova, Ricardo Garcia, Andrew Huang, Fan Wu, Baris Aksanli, and Tajana Rosing. ApproxLP: Approximate multiplication with linearization and iterative error control. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.
[32] Chuangtao Chen, Sen Yang, Weikang Qian, Mohsen Imani, Xunzhao Yin, and Cheng Zhuo. Optimally approximated and unbiased floating-point multiplier with runtime configurability. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.
[33] Chuangtao Chen, Weikang Qian, Mohsen Imani, Xunzhao Yin, and Cheng Zhuo. PAM: A Piecewise-Linearly-Approximated Floating-Point Multiplier with Unbiasedness and Configurability. *IEEE Transactions on Computers*, 2021.

[34] Mohsen Imani, Xunzhao Yin, John Messerly, Saransh Gupta, Michael Niemier, Xiaobo Sharon Hu, and Tajana Rosing. Searchd: A memory-centric hyperdimensional computing with stochastic training. *IEEE TCAD*, 2019.

[35] Kai Ni, Xunzhao Yin, Ann Franchesca Laguna, Siddharth Joshi, Stefan Dünkel, Martin Trentzsch, Johannes Müller, Sven Beyer, Michael Niemier, Xiaobo Sharon Hu, et al. Ferroelectric ternary content-addressable memory for one-shot learning. *Nature Electronics*, 2(11):521–529, 2019.

[36] Shaghayegh Vahdat, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. TOSAM: An energy-efficient truncation- and rounding-based scalable approximate multiplier. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(5):1161–1173, 2019.

[37] Reza Zendegani, Mehdi Kamal, Milad Bahadori, Ali Afzali-Kusha, and Massoud Pedram. RoBA multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(2):393–401, 2016.

[38] Syed Shakib Sarwar, Swagath Venkataramani, Anand Raghunathan, and Kaushik Roy. Multiplier-less artificial neurons exploiting error resiliency for energy-efficient neural computing. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 145–150. IEEE, 2016.

[39] Weiqiang Liu, Jiahua Xu, Danye Wang, Chenghua Wang, Paolo Montuschi, and Fabrizio Lombardi. Design and evaluation of approximate logarithmic multipliers for low power error-tolerant applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(9):2856–2868, 2018.

[40] Mohammad S Ansari, Bruce F Cockburn, and Jie Han. An improved logarithmic multiplier for energy-efficient neural computing. *IEEE Transactions on Computers*, 70(4):614–625, 2020.

[41] Mohammad S Ansari, Bruce F Cockburn, and Jie Han. A hardware-efficient logarithmic multiplier with improved accuracy. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 928–931. IEEE, 2019.

[42] Hassaan Saadat, Haseeb Bokhari, and Sri Parameswaran. Minimally biased multipliers for approximate integer and floating-point multiplication. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2623–2635, 2018.

[43] Soheil Hashemi, R Iris Bahar, and Sherief Reda. DRUM: A dynamic range unbiased multiplier for approximate applications. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 418–425. IEEE, 2015.

[44] Shaghayegh Vahdat, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. LETAM: A low energy truncation-based approximate multiplier. *Computers & Electrical Engineering*, 63:1–17, 2017.

[45] Srinivasan Narayanamoorthy, Hadi Asghari Moghaddam, Zhenhong Liu, Taejoon Park, and Nam Sung Kim. Energy-efficient approximate multiplication for digital signal processing and classification applications. *IEEE transactions on very large scale integration (VLSI) systems*, 23(6):1180–1184, 2014.

[46] Suganthi Venkatachalam and Seok-Bum Ko. Design of power and area efficient approximate multipliers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(5):1782–1786, 2017.

[47] Tongxin Yang, Tomoaki Ukezono, and Toshinori Sato. A low-power high-speed accuracy-controllable approximate multiplier design. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 605–610. IEEE, 2018.

[48] Tongxin Yang, Tomoaki Ukezono, and Toshinori Sato. Low-power and high-speed approximate multiplier design with a tree compressor. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 89–96. IEEE, 2017.

[49] Darjn Esposito, Antonio Giuseppe Maria Strollo, Ettore Napoli, Davide De Caro, and Nicola Petra. Approximate multipliers based on new approximate compressors. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(12):4169–4182, 2018.

[50] Manzhen Wang, Yuanyong Luo, Mengyu An, Yuou Qiu, Muhan Zheng, Zhongfeng Wang, and Hongbing Pan. An Optimized Compression Strategy for Compressor-Based Approximate Multiplier. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020.

[51] Minho Ha and Sunggu Lee. Multipliers with approximate 4–2 compressors and error recovery modules. *IEEE Embedded Systems Letters*, 10(1):6–9, 2017.

[52] Che-Wei Tung and Shih-Hsu Huang. Low-power high-accuracy approximate multiplier using approximate high-order compressors. In *2019 2nd International Conference on Communication Engineering and Technology (ICCET)*, pages 163–167. IEEE, 2019.

[53] Zhixi Yang, Jie Han, and Fabrizio Lombardi. Approximate compressors for error-resilient multiplier design. In *2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, pages 183–186. IEEE, 2015.

[54] Nguyen Van Toan and Jeong-Gun Lee. FPGA-based multi-level approximate multipliers for high-performance error-resilient applications. *IEEE Access*, 8:25481–25497, 2020.

[55] Weiqiang Liu, Liangyu Qian, Chenghua Wang, Honglan Jiang, Jie Han, and Fabrizio Lombardi. Design of approximate radix-4 Booth multipliers for error-tolerant computing. *IEEE Transactions on Computers*, 66(8):1435–1441, 2017.

[56] Honglan Jiang, Jie Han, Fei Qiao, and Fabrizio Lombardi. Approximate radix-8 Booth multipliers for low-power and high-performance operation. *IEEE Transactions on Computers*, 65(8):2638–2644, 2015.
[57] S Pabithra and S Nageswari. Analysis of approximate multiplier using 15–4 compressor for error tolerant application. In *2018 International Conference on Control, Power, Communication and Computing Technologies (ICCPCCT)*, pages 410–415. IEEE, 2018.
[58] Peipei Yin, Chenghua Wang, Weiqiang Liu, Earl E Swartzlander, and Fabrizio Lombardi. Designs of approximate floating-point multipliers with variable accuracy for error-tolerant applications. *Journal of Signal Processing Systems*, 90(4):641–654, 2018.
[59] Sunil Ashtaputre, Carla D Savage, and Wesley E Snyder. Using an approximate multiplier in a one-dimensional array architecture for real-time convolution. Technical report, North Carolina State University. Center for Communications and Signal Processing, 1985.
[60] Avinash Lingamneni, Arindam Basu, Christian Enz, Krishna V Palem, and Christian Piguet. Improving energy gains of inexact DSP hardware through reciprocative error compensation. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–8. IEEE, 2013.
[61] Marcelo Brandalero, Antonio Carlos S Beck, Luigi Carro, and Muhammad Shafique. Approximate on-the-fly coarse-grained reconfigurable acceleration for general-purpose applications. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.
[62] Omid Akbari, Mehdi Kamal, Ali Afzali-Kusha, Massoud Pedram, and Muhammad Shafique. PX-CGRA: Polymorphic approximate coarse-grained reconfigurable architecture. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 413–418. IEEE, 2018.
[63] Byoung-Joo Yoo, Dong-Hyuk Lim, Hyonguk Pang, June-Hee Lee, Seung-Yeob Baek, Naxin Kim, Dong-Ho Choi, Young-Ho Choi, Hyeyeon Yang, Taehun Yoon, et al. 6.4 A 56Gb/s 7.7 mW/Gb/s PAM-4 wireline transceiver in 10nm FinFET using MM-CDR-Based ADC timing skew control and low-power DSP with approximate multiplier. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 122–124. IEEE, 2020.
[64] Farhana Sharmin Snigdha, Deepashree Sengupta, Jiang Hu, and Sachin S Sapatnekar. Optimal design of JPEG hardware under the approximate computing paradigm. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2016.
[65] Zidong Du, Krishna Palem, Avinash Lingamneni, Olivier Temam, Yunji Chen, and Chengyong Wu. Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators. In *2014 19th Asia and South Pacific design automation conference (ASP-DAC)*, pages 201–206. IEEE, 2014.
[66] Qian Zhang, Ting Wang, Ye Tian, Feng Yuan, and Qiang Xu. ApproxANN: An approximate computing framework for artificial neural network. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 701–706. IEEE, 2015.
[67] Vojtech Mrazek, Syed Shakib Sarwar, Lukas Sekanina, Zdenek Vasicek, and Kaushik Roy. Design of power-efficient approximate multipliers for approximate artificial neural networks. In *Proceedings of the 35th International Conference on Computer-Aided Design*, pages 1–7, 2016.
[68] Issam Hammad and Kamal El-Sankary. Impact of approximate multipliers on VGG deep learning network. *IEEE Access*, 6:60438–60444, 2018.
[69] Mohammad S Ansari, Vojtech Mrazek, Bruce F Cockburn, Lukas Sekanina, Zdenek Vasicek, and Jie Han. Improving the accuracy and hardware efficiency of neural networks using approximate multipliers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(2):317–328, 2019.
[70] Issam Hammad, Kamal El-Sankary, and Jason Gu. Deep learning training with simulated approximate multipliers. In *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 47–51. IEEE, 2019.
[71] Vasileios Leon, Konstantinos Asimakopoulos, Sotirios Xydis, Dimitrios Soudris, and Kiamal Pekmestzi. Cooperative arithmetic-aware approximation techniques for energy-efficient multipliers. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.
[72] Issam Hammad, Ling Li, Kamal El-Sankary, and W Martin Snelgrove. CNN inference using a preprocessing precision controller and approximate multipliers with various precisions. *IEEE Access*, 9:7220–7232, 2021.
[73] Ying Wu and Cheng Zhuo. Verilog implementation of approximate multipliers. https://github.com/skycrapers/AM-Lib, 2022.
[74] IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019.
[75] Christopher S Wallace. A suggestion for a fast multiplier. *IEEE Transactions on electronic Computers*, EC-13(1):14–17, 1964.
[76] Luigi Dadda. Some schemes for parallel multipliers. *Alta frequenza*, 34:349–356, 1965.
[77] Chip-Hong Chang, Jiangmin Gu, and Mingyan Zhang. Ultra low-voltage low-power CMOS 4-2 and 5-2 compressors for fast arithmetic circuits. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 51(10):1985–1997, 2004.

[78] Abdoreza Pishvaie, Ghassem Jaberipur, and Ali Jahanian. Improved CMOS (4; 2) compressor designs for parallel multipliers. *Computers & Electrical Engineering*, 38(6):1703–1716, 2012.

[79] Dursun Baran, Mustafa Aktan, and Vojin G Oklobdzija. Energy efficient implementation of parallel CMOS multipliers with improved compressors. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, pages 147–152, 2010.

[80] Armineh Arasteh, Mohammad H Moaiyeri, MohammadReza Taheri, Keivan Navi, and Nader Bagherzadeh. An energy and area efficient 4:2 compressor based on finfets. *Integration*, 60:224–231, 2018.

[81] Sreehari Veeramachaneni, Kirthi M Krishna, Lingamneni Avinash, Sreekanth Reddy Puppala, and MB Srinivas. Novel architectures for high-speed and low-power 3-2, 4-2 and 5-2 compressors. In *20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems (VLSID'07)*, pages 324–329, 2007.

[82] Christopher Fritz and Adly T Fam. Fast binary counters based on symmetric stacking. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2971–2975, 2017.

[83] Amir Momeni, Jie Han, Paolo Montuschi, and Fabrizio Lombardi. Design and analysis of approximate compressors for multiplication. *IEEE Transactions on Computers*, 64(4):984–994, 2014.

[84] Syed E Ahmed, Sanket Kadam, and MB Srinivas. An iterative logarithmic multiplier with improved precision. In *2016 IEEE 23nd Symposium on Computer Arithmetic (ARITH)*, pages 104–111. IEEE, 2016.

[85] Vincent Camus, Jeremy Schlachter, Christian Enz, Michael Gautschi, and Frank 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*, pages 465–468, 2016.

[86] Kartikeya Bhardwaj, Pravin S Mane, and Jörg Henkel. Power- and area-efficient Approximate Wallace Tree Multiplier for error-resilient systems. In *Fifteenth International Symposium on Quality Electronic Design*, pages 263–269, 2014.

[87] Cong Liu, Jie Han, and Fabrizio Lombardi. A low-power, high-performance approximate multiplier with configurable partial error recovery. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–4. IEEE, 2014.

[88] Chuliang Guo, Li Zhang, Xian Zhou, Weikang Qian, and Cheng Zhuo. A reconfigurable approximate multiplier for quantized CNN applications. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 235–240. IEEE, 2020.

[89] Liangyu Qian, Chenghua Wang, Weiqiang Liu, Fabrizio Lombardi, and Jie Han. Design and evaluation of an approximate Wallace-Booth multiplier. In *2016 IEEE international symposium on circuits and systems (ISCAS)*, pages 1974–1977. IEEE, 2016.

[90] Avinash Lingamneni, Christian Enz, Jean-Luc Nagel, Krishna Palem, and Christian Piguet. Energy parsimonious circuit design through probabilistic pruning. In *2011 Design, Automation & Test in Europe*, pages 1–6. IEEE, 2011.

[91] Jeremy Schlachter, Vincent Camus, Christian Enz, and Krishna V Palem. Automatic generation of inexact digital circuits by gate-level pruning. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 173–176. IEEE, 2015.

[92] Mohsen Imani, Daniel Peroni, and Tajana Rosing. CFPU: Configurable floating point multiplier for energy-efficient computing. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2017.

[93] Mohsen Imani, Ricardo Garcia, Saransh Gupta, and Tajana Rosing. Rmac: Runtime configurable floating point multiplier for approximate computing. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 1–6, 2018.

[94] Honglan Jiang, Cong Liu, Fabrizio Lombardi, and Jie Han. Low-power approximate unsigned multipliers with configurable error recovery. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(1):189–202, 2018.

[95] Weihua Xiao, Cheng Zhuo, and Weikang Qian. OPACT: Optimization of approximate compressor tree for approximate multiplier. In *2022 Design, Automation, and Test in Europe Conference (DATE)*, 2022.

[96] Vojtech Mrazek, Radek Hrbacek, Zdenek Vasicek, and Lukas Sekanina. Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 258–261. IEEE, 2017.

[97] Radek Hrbacek, Vojtech Mrazek, and Zdenek Vasicek. Automatic design of approximate circuits by means of multi-objective evolutionary algorithms. In *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–6. IEEE, 2016.

[98] Farzad Farshchi, Muhammad S Abrishami, and Sied M Fakhraie. New approximate multiplier for low power digital signal processing. In *The 17th CSI International Symposium on Computer Architecture & Digital Systems (CADS 2013)*, pages 25–30. IEEE, 2013.

[99] Jienan Chen and Jianhao Hu. Energy-efficient digital signal processing via voltage-overscaling-based residue number system. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(7):1322–1332, 2012.

[100] Yuan-Ho Chen and Tsin-Yuan Chang. A high-accuracy adaptive conditional-probability estimator for fixed-width booth multipliers. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 59(3):594–603, 2011.

[101] Hamid Reza Mahdiani, Ali Ahmadi, Sied Mehdi Fakhraie, and Caro Lucas. Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(4):850–862, 2009.

[102] Mark SK Lau, Keck-Voon Ling, and Yun-Chung Chu. Energy-aware probabilistic multiplier: design and analysis. In *Proceedings of the 2009 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 281–290, 2009.

[103] Jiun-Ping Wang, Shiann-Rong Kuang, and Shish-Chang Liang. High-accuracy fixed-width modified Booth multipliers for lossy applications. *IEEE transactions on very large scale integration (VLSI) systems*, 19(1):52–60, 2009.

[104] Min-An Song, Lan-Da Van, and Sy-Yen Kuo. Adaptive low-error fixed-width Booth multipliers. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 90(6):1180–1187, 2007.

[105] Weiqiang Liu, Jiahua Xu, Danye Wang, and Fabrizio Lombardi. Design of approximate logarithmic multipliers. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pages 47–52, 2017.

[106] Chuangtao Chen, Weikang Qian, Mohsen Imani, Xunzhao Yin, and Cheng Zhuo. Pam: A piecewise-linearly-approximated floating-point multiplier with unbiasedness and configurability. *IEEE Transactions on Computers*, 2021.

[107] Edwin de Angel and EE Swartzlander. Low power parallel multipliers. In *VLSI Signal Processing, Ix*, pages 199–208. IEEE, 1996.

[108] Chia-Hao Lin and Chao Lin. High accuracy approximate multiplier with error correction. In *International Conference on Computer Design*, pages 33–38, 2013.

[109] Omid Akbari, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. Dual-quality 4:2 compressors for utilizing in dynamic accuracy configurable multipliers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(4):1352–1361, 2017.

[110] Mohammad Ahmadinejad, Mohammad H Moaiyeri, and Farnaz Sabetzadeh. Energy and area efficient imprecise compressors for approximate multiplication at nanoscale. *AEU-International Journal of Electronics and Communications*, 110:152859, 2019.

[111] Antonio G. M. Strollo, Ettore Napoli, Davide De Caro, Nicola Petra, and Gennaro Di Meo. Comparison and extension of approximate 4-2 compressors for low-power approximate multipliers. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(9):3021–3034, 2020.

[112] R Marimuthu, Y Elsie Rezinold, and Partha Sharathi Mallick. Design and analysis of multiplier using approximate 15-4 compressor. *IEEE Access*, 5:1027–1036, 2016.

[113] Xuan Wang and Weikang Qian. MinAC: Minimal-area approximate compressor design based on exact synthesis for approximate multipliers. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS), to be published*, 2022.

[114] Winston Haaswijk, Mathias Soeken, Alan Mishchenko, and Giovanni De Micheli. SAT-Based Exact Synthesis: Encodings, Topology Families, and Parallelism. *IEEE TCAD*, 39(4):871–884, 2020.

[115] Honglan Jiang, Cong Liu, Fabrizio Lombardi, and Jie Han. Low-Power Approximate Unsigned Multipliers With Configurable Error Recovery. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66:189–202, 2019.

[116] Hsin-Lei Lin, Robert C Chang, and Ming-Tsai Chan. Design of a novel radix-4 booth multiplier. In *The 2004 IEEE Asia-Pacific Conference on Circuits and Systems*, volume 2, pages 837–840. Citeseer, 2004.

[117] Lukas Sekanina and Zdenek Vasicek. Approximate circuit design by means of evolvable hardware. In *2013 IEEE International Conference on Evolvable Systems (ICES)*, pages 21–28. IEEE, 2013.

[118] Zdenek Vasicek and Lukas Sekanina. Evolutionary design of approximate multipliers under different error metrics. In *17th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*, pages 135–140. IEEE, 2014.

[119] Julian Francis Miller and Simon L Harding. Cartesian genetic programming. In *Proceedings of the 10th annual conference companion on Genetic and evolutionary computation*, pages 2701–2726, 2008.

[120] Debabrata Mohapatra, Vinay K Chippa, Anand Raghunathan, and Kaushik Roy. Design of voltage-scalable meta-functions for approximate computing. In *2011 Design, Automation & Test in Europe*, pages 1–6. IEEE, 2011.

[121] Synopsys. Design compiler. https://www.synopsys.com/, 2022.

[122] UMC. Umc40. https://www.umc.com, 2022.

[123] Wilson Snyder. Verilator. https://github.com/verilator/verilator, 2003-2022.

[124] Synopsys. Designware. https://www.synopsys.com/designware-ip.html, 2022.

[125] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[126] Xian Zhou, Li Zhang, Chuliang Guo, Xunzhao Yin, and Cheng Zhuo. A convolutional neural network accelerator architecture with fine-granular mixed precision configurability. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2020.