# Approximate Multiplier Design for Energy Efficiency: From Circuit to Algorithm*

Ying Wu, Chuangtao Chen, Chenyi Wen, Weikang Qian,
Xunzhao Yin, Cheng Zhuo

**Abstract** Due to the inherent tolerance to inaccuracy in data-driven applications, approximate multipliers have received growing attentions for its compactness and energy-efficiency. However, the energy efficiency of an approximate multiplier largely depends on how and where the inaccuracy is introduced into the design, which can be roughly categorized to 3 design levels: (1) architecture, (2) algorithm, and (3) circuit. Such large design space inevitably incurs design complexities and challenges in selecting the appropriate multiplier for a particular application. Thus, this paper provides a comprehensive review of the state-of-the-art (SOTA) designs of approximate multipliers for future investigations.

Ying Wu
College of Information Science and Electronic Engineering, Zhejiang University, China. e-mail: ying.wu@zju.edu.cn

Chuangtao Chen
College of Electrical Engineering, Zhejiang University, China. e-mail: chtchen@zju.edu.cn

Chenyi Wen
College of Information Science and Electronic Engineering, Zhejiang University, China. e-mail: wwency@zju.edu.cn

Weikang Qian
University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, China. e-mail: qianwk@sjtu.edu.cn

Xunzhao Yin
College of Information Science and Electronic Engineering, Zhejiang University, China. e-mail: xzyin1@zju.edu.cn

Cheng Zhuo
College of Information Science and Electronic Engineering, Zhejiang University, China. e-mail: czhuo@zju.edu.cn

* This work was partially supported by National Key R&D Program of China (Grant No. 2018YFE0126300) and National Natural Science Foundation of China (Grant No. 62034007 and 61974133). Corresponding author: Cheng Zhuo.

## 1 Introduction

Thanks to the rapid growth of Artificial Intelligence (AI) and Internet-of-Things (IoT), energy efficiency has become a critical concern for IoT devices with constrained resources [1]. Among various efforts for energy efficiency optimization, approximate computing has emerged as a promising alternative for designers to trade computational accuracy with energy efficiency. This is especially applicable to human sensory or machine learning tasks where a small amount of inaccuracy is tolerable [2–6].

At the edge, IoT devices are designed to consume the minimum resource to achieve the desired accuracy. However, the conventional processors, such as CPU or GPU, can only conduct all the computations with pre-determined but sometimes unnecessary precisions, inevitably degrading their energy efficiency. For example, when running data-intensive applications, *e.g.*, streaming, neural network, and image processing, *etc.*, multiplication is frequently invoked and consumes non-trivial energy [7]. However, for a neural network, even with an inaccurate multiplier with limited precision, such inaccuracy may get cancelled out without impacting the inference accuracy [8]. In other words, when running inaccuracy-tolerable applications on the conventional processors, significant energy and time are actually spent on the multipliers computing highly accurate outputs that are not necessarily demanded. Thus, for the multiplication in IoT devices, there is a need to optimize its energy efficiency by providing sufficient instead of excessively accurate computational precisions.

As a common arithmetic component that has been studied for decades [9, 10], the past focus for the multiplier is mainly placed upon accuracy and performance. Recently, with awareness of the compromise between the stringent resource constraint and the accuracy tolerance for edge applications, there have been various research efforts for approximate multiplier design and optimization, ranging from algorithm, architecture, to circuit [11–25].

Since many prior designs happen to rely on hand-crafted structures or heuristics, it is then highly desired to systematically review and understand the pros and cons of how and where the inaccuracy can be introduced into the design. Thus, this chapter will review approximate multipliers from three different levels, *i.e.*, architecture, algorithm, and circuit. The remainder of this chapter is organized as follows. In section 2, we review the background of approximate multiplier. Sections 3 to 5 discuss the approximate multipliers at architecture, algorithm, and circuit levels, respectively, followed by the conclusions in section 6.

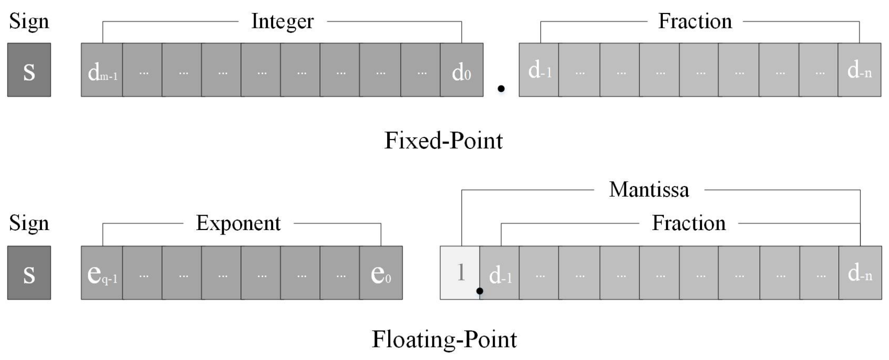


**Fig. 1** Comparison between fixed-point and floating-point formats.

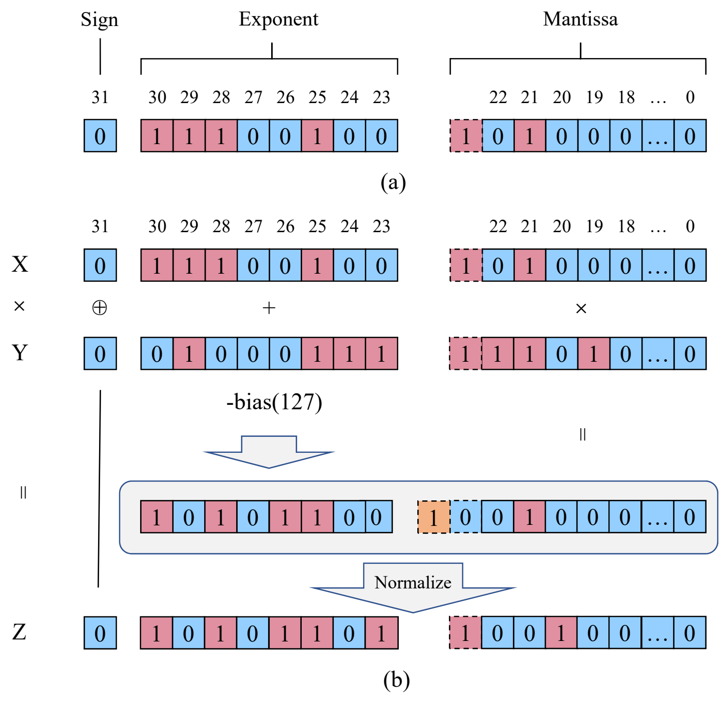


**Fig. 2** An example of a 32-bit floating point multiplication according to IEEE 754 standard.

## 2 Background

### 2.1 Fixed Point v.s. Floating Point

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 is whether the implementation has a specific number of digits reserved for the integer and fractional parts, respectively. In other words, fixed point numbers have a decimal point at a fixed position. Obviously, floating point 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 fixed point and 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 [26], which is a technical standard for floating point arithmetic, a floating point number consists of sign, exponent, and mantissa. The mantissa of a *normalized* floating point number is a fraction with its value between 1 and 2, where its first digit is fixed to 1 and the rest is the fraction in the range of [0,1).

Depending on the underlying number representations, designers may use either a fixed point multiplier or a floating point multiplier to conduct multiplication. For the floating point numbers, the multiplication procedure of 32-bit floating point numbers is demonstrated in Fig. 2. The sign bits are XORed together and the exponents are summed by an adder. Then, a bias of $2^{exponent\_width} - 1$ 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 mantissa part is much more energy- and delay-consuming than the other two parts, which is hence the focus of most research work [24, 25]. On the other hand, if no overflow, the fixed point multiplication is carried out as a regular multiplication with its fractional part truncated to the designed bit-width. However, the difference between the two multiplications is actually smaller than it seems. The multiplication of the mantissa parts for 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.

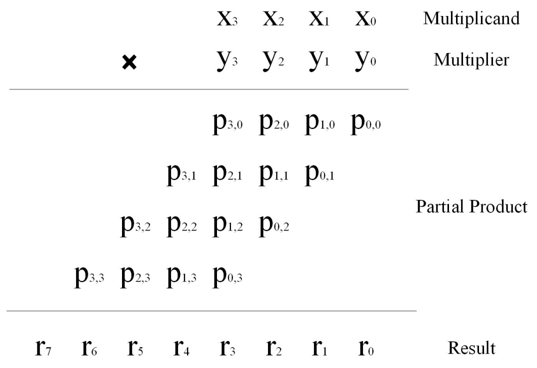
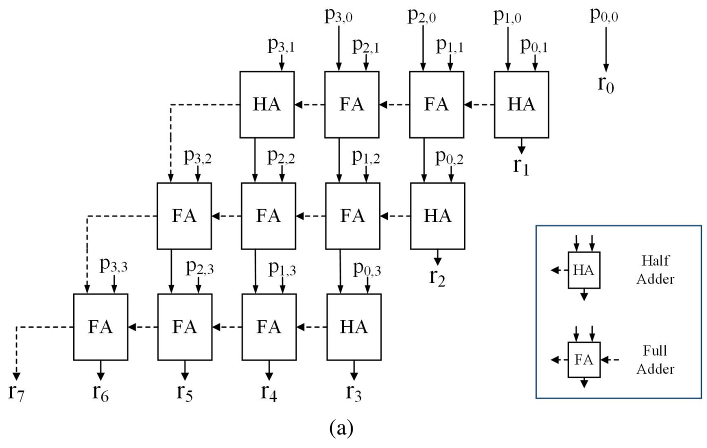

$x_3$ $x_2$ $x_1$ $x_0$
Multiplicand
$\times$ $y_3$ $y_2$ $y_1$ $y_0$
Multiplier
$p_{3,0}$ $p_{2,0}$ $p_{1,0}$ $p_{0,0}$
$p_{3,1}$ $p_{2,1}$ $p_{1,1}$ $p_{0,1}$
$p_{3,2}$ $p_{2,2}$ $p_{1,2}$ $p_{0,2}$
$p_{3,3}$ $p_{2,3}$ $p_{1,3}$ $p_{0,3}$
Partial Product
$r_7$ $r_6$ $r_5$ $r_4$ $r_3$ $r_2$ $r_1$ $r_0$
Result

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



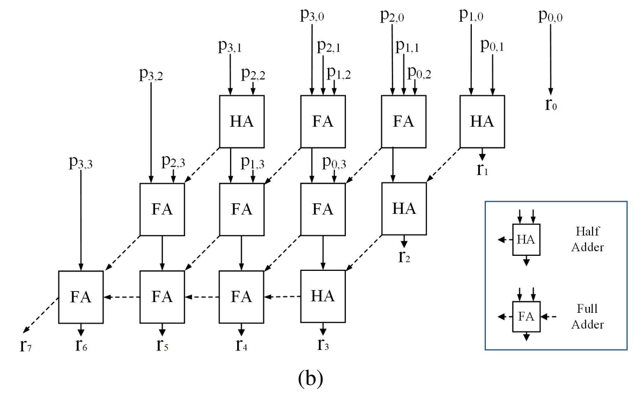


**Fig. 4** (a) An example of RSA based accumulator; (b) An example of CSA based accumulator.

## 2.2 Binary Multiplier

Before we go into details of approximate multiplier design, we would like to introduce the basics of a binary multiplier, which is straightforward but sheds light on different approximation techniques introduced in the latter sections.

Due to the nature of dealing with only two digits, *i.e.*, 0 and 1, binary multiplication actually can be considered as a process of addition and shifting. For example, assume we have two 4-bit operands of $x$ and $y$, where $x$ is a multiplicand and $y$ is a multiplier. As shown in Fig. 3, similar as a decimal multiplication operation, 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

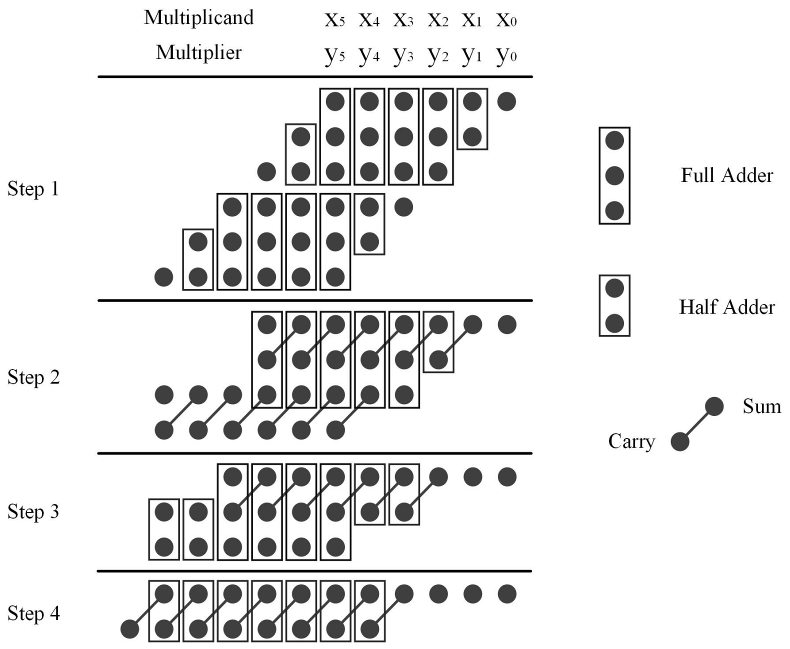


**Fig. 5** An example of Wallace Tree based multiplier.

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 multiplication result of $\{r_{7,0}, r_{6,0}, r_{5,0}, r_{4,0}, r_{3,0}, r_{2,0}, r_{1,0}, r_{0,0}\}$. Thus, the operation of a binary multiplier can be roughly divided to three stages, data input, partial product generation, and accumulation.

Since the binary product does not generate 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 partial products as shown in Fig. 4(a), where HA refers to half adder and FA refers to full adder. Obviously, the critical path of such a structure is the carry propagation. For example, Fig. 4(a) demonstrates a ripple-carry adder (RCA) based accumulator, with the carries propagated horizontally from right to left, while Fig. 4(b) plots 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 accumulation, C. S. Wallace proposed the Wallace Tree structure in 1964 [27]. As shown in Fig. 5, the Wallace Tree groups three partial products together column-wisely 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, *i.e.*, 4 steps as in Fig. 5, which are then 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 [27].

### 2.3 Approximate Multiplier

Approximate arithmetic has been a popular research area in the past decade. Many prior work on approximate multiplier tackle the problem by introducing approximations at circuit, architecture or algorithmic levels to reduce critical path delay or improve energy efficiency. For example, references [17,28–34] propose to approximate K-map or prune out a few gates to simplify the gate netlist. Many work also focused on improving the conventional multiplier architecture with approximate components, such as adders, to speed up addition or partial product generation [17,35–40]. Kulkarni *et al.* proposed to construct a new approximate multiplier architecture using a modified 2×2 multiply block [17]. From an even higher design level, Ahmed *et al.* proposed a pipelined log-based approximation using the classical Mitchell multiplier with an iterative procedure to improve the accuracy [41]. To speed up the iterative procedure, they proposed to truncate the bits after the leading one to save energy. To satisfy various accuracy requirement in different scenarios, another alternative is to utilize hybrid methods with both approximate and accurate multipliers to adjust the computational accuracy by selecting the appropriate multiplier, thereby trading off between accuracy and cost [42,43].

For all the prior work with various approximation techniques, it is actually very challenging to precisely categorize the introduced approximation to a particular design level, *i.e.*, circuit, architecture, or algorithm. Many of them actually involve multiple design levels, as the high level approximation, *e.g.*, algorithm, may always incur additional architecture changes [24,25,41–43].

In order to facilitate our review in the following sections, we would like to utilize the following rules for categorization:

- 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.
- Algorithm: The introduced approximation originates from a different algorithm to conduct multiplication.
- Circuit: The approximation technique is not limited to a particular multiplier architecture/algorithm and can be combined with the approximation techniques at other design levels.

## 3 Approximate Multiplier with Architecture Level Approximation

As is discussed in section 2, the conventional multiplier typically involves three stages, *i.e.*, data input, partial product generation, and accumulation. To reduce the number of partial products, an encoding stage can be included, *e.g.*, Booth encoding [44]. When we design approximate multipliers based on such an architecture, approximations can be introduced into any of the four aforementioned stages.

### 3.1 Approximation at Input

It is simple yet effective to introduce approximation in data input for approximate multiplier design. For example, we can remove a few least significant bits (LSBs) of the input to reduce the input bit-width, which is supposed to have lower impact on the result than those most significant bits (MSBs) [36, 45–47]. In general, there are two types of data segmentation, dynamic segment method (DSM) and static segment method (SSM) [36]. DSM segments the data according to the leading one, while SSM is based on a given segmentation option. For example, as shown in Fig. 6(a), DSM keeps $k$ consecutive bits from the first non-zero bit of an unsigned number. The parameter $k$ determines the level of accuracy loss for approximate multiplier. On the other hand, SSM in Fig. 6(b) provides a few pre-determined ($i.e.$, *static*) options when truncating the input data. The options can be like either leading $k$ bits (option 1) or last $k$ bits (option 2), as suggested in [36]. It is also possible to keep the bits in the middle (option 3) as a trade-off (Fig. 6(b)). Unlike DSM, SSM consumes less hardware resources but may include more redundant bits. In [45], the additional support for DSM requires 2 extra Leading-One Detectors (LOD), 2 extra encoders and 1 extra barrel shifter.

Table 1 compares several approximate multipliers with approximation at input stage, the results of which are compiled from [36, 45–47]. The data is collected on 8-bit unsigned multiplication using 45-nm Nangate technology. Five approximate multipliers are included here and compared to an accurate multiplier [36, 45–47], where SSM [36] is the approximate multiplier using SSM to truncate the input data; DSM [36] and DRUM [45] both use DSM to truncate, while DRUM [45] always sets the last bit to 1 and DSM [36] leaves as it is; LETAM [47] and TOSAM [46] truncate both partial product and bit-width, while TOSAM uses 2 separate parameters for partial product and bit-width, respectively, and LETAM only uses 1 parameter for both. Parameter $k$ is the number in the brackets for each multiplier. Five metrics are presented in the table, mean relative error distance (MRED), power, delay, area, and power-delay product (PDP) as the metric for energy efficiency. As shown in the table, SSM results in a smaller area and power while the other DSM based methods consume at least 1.5× larger area. Moreover, a larger bit-width or more complex control generally yields to a higher accuracy or smaller delay but at the cost of larger power and area consumption. Among all the DSM based multipliers, DRUM [45] provides a better trade-off between accuracy and energy efficiency.

### 3.2 Approximation at Partial Product Generation

Venkatachalam *et al.* proposes an under-designed multiplier (UDM) architecture, which brings approximation into the partial product generation stage [17]. UDM partitions both multiplier and multiplicand into 2 parts, and then formulates a $2 \times 2$ multiplication. As shown in Fig. 7, each partial product can be produced with an approximate multiplier. Another alternative to partial product generation is to

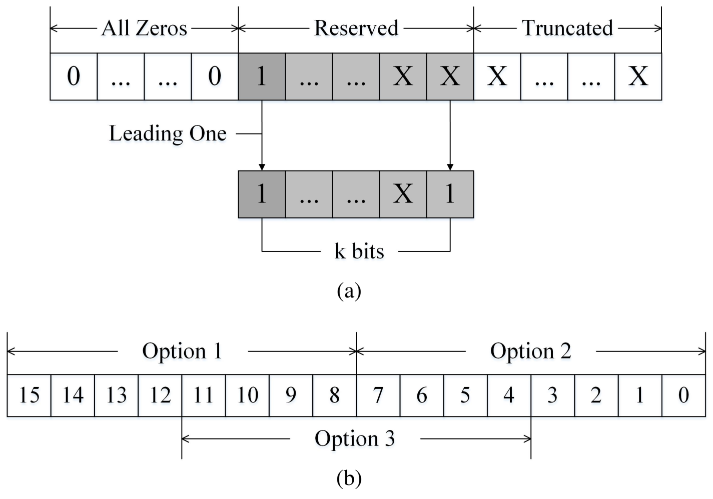


**Fig. 6** (a) An example of DSM truncation; (b) An example of SSM truncation.

**Table 1** Comparison on 8-bit approximation multipliers with input approximations [46].

| Multiplier | MRED | power (uW) | Delay (ns) | Area (um$^2$) | PDP (fJ) |
|---|---|---|---|---|---|
| Accurate | 0 | 360 | 0.85 | 417 | 306 |
| SSM [36] | N/A | 68 | N/A | 75 | N/A |
| DSM(3) [36] | 0.1444 | 128 | 0.8 | 182 | 102.2 |
| DSM(4) [36] | 0.0680 | 205 | 1.08 | 233 | 221.46 |
| DRUM(3) [45] | 0.1260 | 104 | 0.7 | 143 | 72.73 |
| DRUM(4) [45] | 0.0640 | 172 | 1 | 208 | 172.16 |
| LETAM(3) [47] | 0.0290 | 270 | 1 | 310 | 270 |
| TOSAM(1,5) [46] | 0.0406 | 231 | 0.88 | 291 | 203.44 |

introduce an intermediate variable to replace the partial products ($a.k.a.$ altered partial product (APP)) and then conduct approximations [48–50]. As discussed in section 2, a partial product can be generated using AND gates:

$$pp_{m,n} = x_m \cdot y_n \ , \tag{1}$$

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

$$p_{m,n} = pp_{m,n} + pp_{n,m} \ , \tag{2}$$

$$g_{m,n} = pp_{m,n} \cdot pp_{n,m} \ . \tag{3}$$

Since the generate signals are possibly all 0's, they can then be compressed column-wisely using an OR gate. The propagate signals can be computed with approximate adders to achieve a more compact design than the original multiplier. Yang *et al.*

employs a similar idea of using two signals of approximate sum and error recovery vector to approximate the partial product [49]. Table 2 compares the impact of different partial product approximation methods with results compiled from [17,48], where UDM refers to the method in Fig. 7 [17]; APP and APP_M refer to the approximate multiplier using altered partial products as in [48], while the most significant column is accurately computed without approximation in APP_M. The designs are compared on MRED, normalized mean error distance (NMED), power, delay, area, and PDP. It is expected that APP incurs larger error than APP_M with smaller area, delay, and power. UDM is more accurate than APP at the cost of more than 2× power consumption, but inferior to APP_M almost across all the metrics. Thus, APP_M achieves a better trade-off between accuracy and energy efficiency.

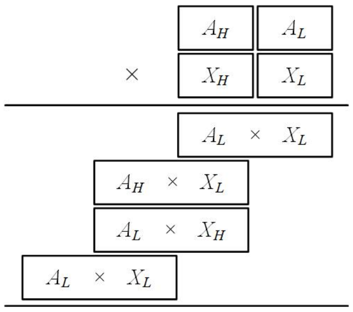


**Fig. 7** An example of UDM in [17].

**Table 2** Comparison on 16-bit approximate multipliers using partial product approximation [48].

| Multiplier | MRED | NMED | Power (uW) | Delay (ns) | Area (um$^2$) | PDP (fJ) |
|---|---|---|---|---|---|---|
| Accurate | 0 | 0 | 1776.49 | 0.68 | 4859.28 | 1208.01 |
| UDM [17] | 3.32e-2 | 1.39e-2 | 1318.51 | 0.67 | 3938 | 883.4 |
| APP [48] | 7.63e-2 | 1.78e-2 | 503.15 | 0.47 | 2158.56 | 236.48 |
| APP_M [48] | 2.44e-4 | 7.10e-6 | 1102.03 | 0.66 | 3319.2 | 727.34 |

## 3.3 Approximation at Accumulation

At accumulation stage, adders and compressors are the major computing modules. In addition to approximate adders, it is a natural idea to use approximate compressors to speed up accumulation [30, 38, 48–53]. For example, Venkatachalam *et al.* use approximate adders and approximate compressors to compress the partial product array to two rows, which are then added up through a ripple carry adder [48]. Liu *et al*. further propose to ignore the carry signals in adders to reduce the critical path delay, which are then utilized later for error recovery [38].

Recently, many researchers also propose to separate the partial product array column-wisely to two or three groups, as shown in Fig. 8. As the leading bits may have larger impact on accuracy, each group can introduce different levels of approximation [30, 49, 50, 52, 53]. For example, OR gates can be deployed in the last group for LSBs to reduce the hardware cost. The accuracy of the approximate multiplier can be tuned by adjusting the control parameters [49]. References [30, 52] propose to use high-order approximate compressors with error recovery for the partial product groups with lower accuracy requirements. It is also straightforward to completely ignore the less important groups of partial products as more aggressive approximation. Mahdiani *et al*. propose to divide the partial product array into four groups through horizontal and vertical slicing, as shown in Fig. 9 [54]. 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.

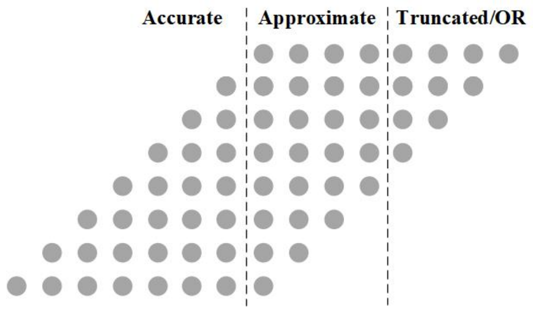


**Fig. 8** An example of partial product array that is divided into three groups with different levels of approximation.

## 3.4 Approximate at Booth Encoding

Booth encoding is used to reduce the number of partial products, which can be generated in parallel at the cost of additional area. Radix-4 Booth algorithm is a common option deployed for high-bit-width multipliers [55]. Qian *et al*. propose an approximate Wallace-Booth multiplier with approximate modified Booth encoding (MBE), approximate 4-2 compressors, and approximate Wallace tree [31]. In addition to Radix-4 algorithm, Radix-8 Booth algorithm is also widely used to further reduce the number of partial products. However, Radix-8 algorithm demands odd multiples and hence needs additional adders. To reduce the increased partial product generation

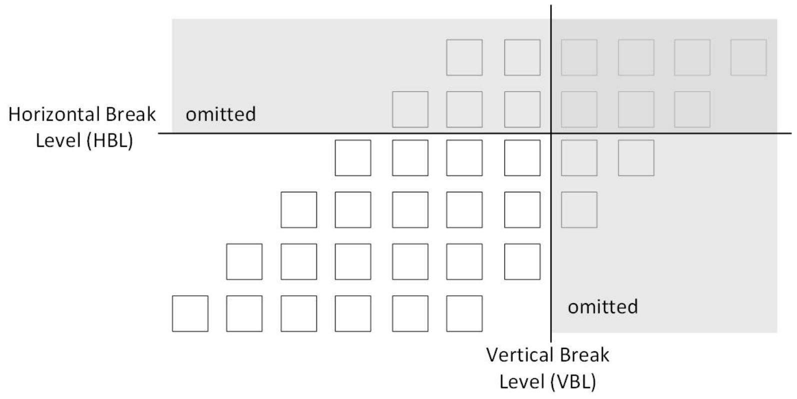


**Fig. 9** An example of Broken-Array Multiplier (BAM) [54].

delay in Radix-8 algorithm, Jiang *et al.* suggest 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.

## 4 Approximate Multiplier with Algorithm Level Approximation

Unlike the work in the last section that modify the reference multiplier architecture, some researchers propose to rebuild the multiplication operation from a higher level, $i.e.$, algorithm, which naturally results in a new multiplier architecture. In this section we will review three different multiplier approximations at 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 *et al.* in 1962 [57]. For a multiplication of $A \times B$, we have:

$$A = 2^{k_1}(1 + x_1) \; , \tag{4}$$

$$\log_2(A) = k_1 + \log_2(1 + x_1) \; , \tag{5}$$

where $A$ is the input operand, $k_1$ is the position of leading one, and $x_1$ is the fraction part 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 multiplication can be written as:

$$\log_2(A \times B) = k_1 + k_2 + \log_2(1 + x_1) + \log_2(1 + x_2)\ . \tag{6}$$

According to Eq. (6), the implementation based on the Mitchell's algorithm [57] requires leading one detector (LOD), binary-logarithm converter (BLC), adder, and logarithm-binary converter (LBC). The procedure for the Mitchell's algorithm is demonstrated in Fig. 10 for a 16×16 multiplier. To reduce the implementation complexity, the logarithm computation in Eq. (6) can be approximated by:

$$\log_2(x + 1) \approx x, 0 \le x < 1\ . \tag{7}$$

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. 7 can be further approximated as:

$$A \times B \approx \begin{cases} 2^{k_1+k_2}(x_1 + x_2 + 1), & x_1 + x_2 < 1\ , \\ 2^{k_1+k_2+1}(x_1 + x_2), & x_1 + x_2 \ge 1\ . \end{cases} \tag{8}$$

Compared with the original multiplication, when $x_1 + x_2 < 1$, the error of Eq. (8) can be expressed as:

$$\begin{aligned} Error &= A \times B - 2^{k_1+k_2}(x_1 + x_2 + 1) \\ &= 2^{k_1+k_2}(1 + x_1)(1 + x_2) - 2^{k_1+k_2}(x_1 + x_2 + 1) \\ &= 2^{k_1+k_2}x_1x_2 \end{aligned} \tag{9}$$

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 higher accuracy using logarithm-based approximation. In [41], the iterative approximation for $x_1 + x_2 \ge 1$ has been explored together with a truncation scheme. Liu *et al.* further investigate the logarithmic based approximate multipliers using different approximate adders and find that set-one-adder (SOA) can achieve a higher accuracy [58]. As shown in Fig. 11, 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 $m$ 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 logarithmic based approximate multiplier, as the Mitchell's algorithm always underestimates the multiplication result. Similar compensation schemes have been introduced in [59–61] to improve the average error introduced by the Mitchell's algorithm at the cost of area and power consumption.

Table 3 evaluates $8 \times 8$ logarithm-based approximate multipliers using ST Micro's 28nm technology [61]. The results are compiled from [61] to compare the metrics of MRED, NMED, power, delay, area, and PDP. Three logarithm-based approximate multipliers are compared with an accurate Wallace-tree based multiplier, where Mitchell [57] refers to the original Mitchell's algorithm; ALM-SOA-5

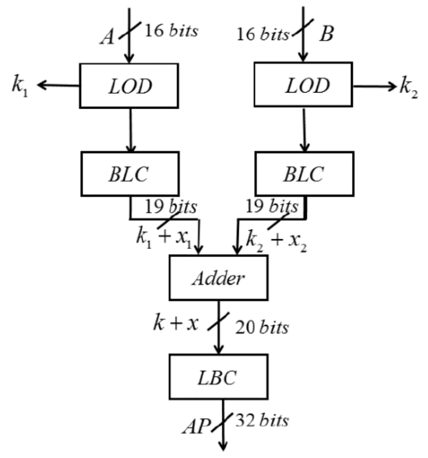


**Fig. 10** Procedure for the Mitchell's algorithm [57].

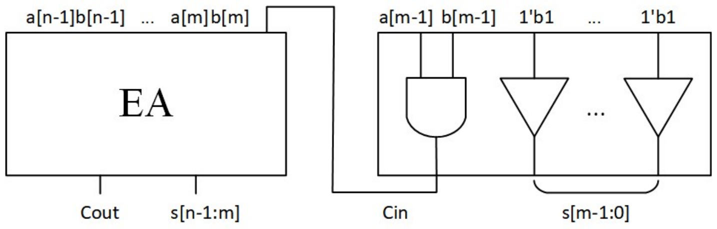


**Fig. 11** Architecture of an $n$-bit set-one-adder [58].

**Table 3** Comparison on 8-bit logarithm based approximate multipliers [61].

| Multipliers | MRED | NMED | Power (uW) | Delay (ns) | Area ($um^2$) | PDP (fJ) |
|---|---|---|---|---|---|---|
| Accurate | 0 | 0 | 99.3 | 1.06 | 235.9 | 105.2 |
| Mitchell [57] | 0.0368 | 0.0014 | 66.26 | 1.42 | 281.2 | 94.09 |
| ALM-SOA-5 [58] | 0.0396 | 0.0007 | 61.04 | 1.39 | 255.4 | 84.84 |
| ILM-5 [61] | 0.0951 | 0.001 | 50.37 | 1.64 | 255.3 | 82.61 |

combines Mitchell's algorithm with SOA with $m = 5$ as in [58]; ILM-5 includes additional rounding of the inputs to the nearest power of 2 on top of the approximation techniques used in ALM-SOA-5. As shown in the table, all the logarithmic approximate multipliers can achieve good accuracy with 33.3-49.3% power reduction consumption, while the delay is increased by 31.1-54.7%. Among the three approximate multipliers, ALM-SOA-5 can achieve a better trade-off between accuracy and energy efficiency.

## 4.2 Approximation with Linearization

Multiplication is a nonlinear operation implemented with a few 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 [24, 25]. It is noted that, while logarithm based approximate designs are built on top of Eq. (8), it is actually a special case of linearization approximation.

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

$$f = xy \approx f_{approx} = ax + by + c;, \tag{10}$$

where $x$ and $y$ are the input operands; $a$, $b$, and $c$ are the coefficients. In [24], an iterative linear approximation for floating-point multiplication is proposed to approximate the multiplication according to Eq. (10). For the mantissas of normalized floating-point numbers, the range 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 12 summarizes the computation procedure called ApproxLP using the linear approximation in [24]. It is clear that the accuracy can be improved by partitioning more sub-domains, the number of which grows exponentially with the approximate level. Thus, the efficiency of ApproxLP in [24] actually quickly degrades with a larger approximation levels. Moreover, the comparators used for each level in Fig. 12 also introduce non-trivial delay overhead. Fig. 13 plots the error distributions of ApproxLP for different approximation levels, which are symmetric over 0 and hence result in a zero average error.

To reduce the number of comparators, Chen *et al.* propose to partition the input domain into identical smaller square sub-domains [25]. For one level higher, each domain (or sub-domains) 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 [25]:

$$\begin{cases} a = \dfrac{y_1 + y_2}{2} \\ b = \dfrac{x_1 + x_2}{2} \\ c = -ab \end{cases} \tag{11}$$

Fig. 14 demonstrates the multi-level approximate multiplier architecture of OAM in [25]. 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 [24], the

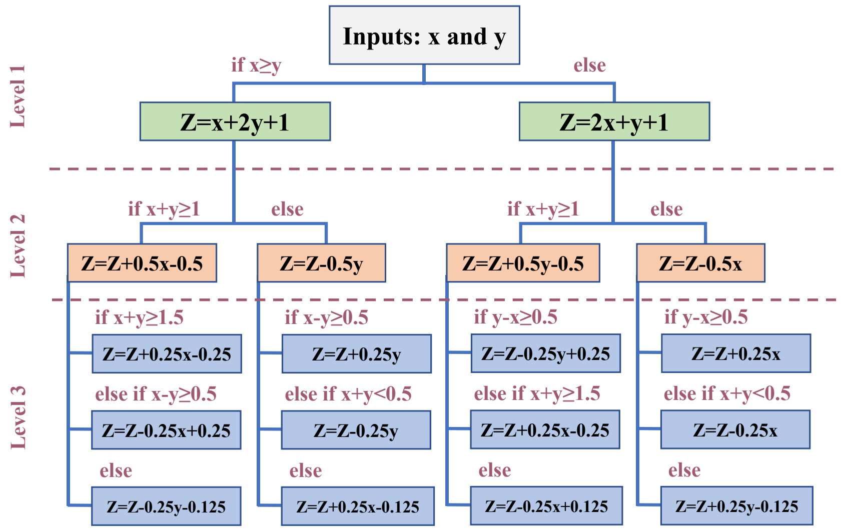


**Fig. 12** Computation procedure using the linear approximation in [24].

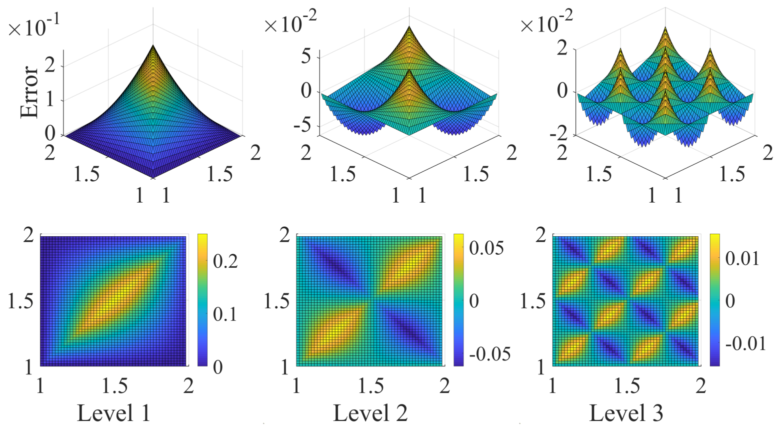


**Fig. 13** Error distributions of ApproxLP at different approximation levels [24].

comparators are no longer needed for OAM [25]. Thus, the delay of OAM can be significantly improved 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 belongs to, a circuit-friendly implementation can be achieved for the error compensation at each level as in Eq. (12) [25]:

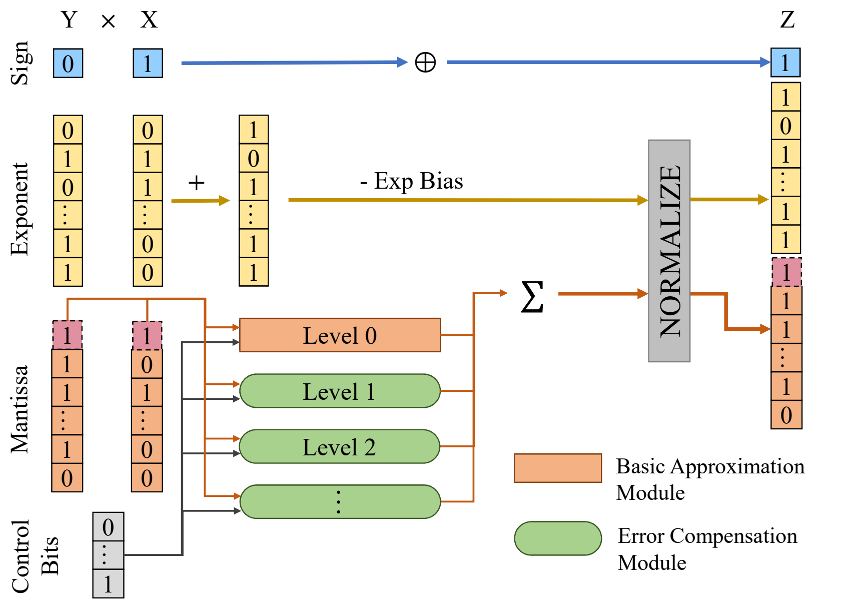


**Fig. 14** Architecture of the approximate multiplier OAM in [25].

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

where "? :" is the conditional operator and $x[n]$ is the $n^{th}$ bit of mantissa $x$; $\oplus$ is XOR operation; $\hat{x_{n-1}}$ is the $n-1$ bits truncation of mantissa $x$ with an extra bit 1 at $n^{th}$ position; $\gg$ represents right shift operation. Since the amount of right shift is pre-determined at each level, the right shift operation does not require additional circuits to implement. Thus, the number of operations at each level for OAM is reduced to 5, which results in a constant area complexity, while ApproxLP has an area complexity of $O(4^n)$ [25].

The errors of the approximate multiplier OAM [25] were reported as below for Maximum-Absolute-Error (MAE), Mean-Square-Error (MSE), and Mean-Absolute-Error (MeanAE) for approximation level $n$:

$$\begin{cases} MAE = \dfrac{1}{4^{n+1}} \\ MSE = \dfrac{1}{9\times 16^{n+1}} \\ MeanAE = \dfrac{1}{4^{n+2}} \end{cases} \tag{13}$$

Similar as ApproxLP [24], OAM [25] has zero-mean error distribution, which is an appealing feature for applications with consecutive multiply-accumulate operations.

Table 4 compares two approximate multipliers using linearization based approximations, where the accurate reference multiplier is a 32-bit floating point multiplier IP from the UMC 40nm library, and the approximate multipliers are configured to different approximation levels as indicated by the numbers in the brackets. The results are compared on MSE, delay, area, and area-delay-product (ADP) as energy efficiency metric. It is found that, for the same approximation level, OAM [25] always performs better than ApproxLP [24]. When compared to the accurate multiplier IP, OAM [25] can achieve 68.6% area saving and 50% delay improvement at the cost of $2.7\times10^{-5}$ MSE, with more than one order of magnitude energy efficiency improvement.

**Table 4** Comparison on approximate multipliers using linearization based approximation.

| Multiplier | MSE | Delay (ns) | Area (um$^2$) | ADP (ns$\cdot$um$^2$) |
|---|---|---|---|---|
| Accurate | 0 | 5.4 | 8219 | 44382.6 |
| ApproxLP (0) | N/A | 2.0 | 1446 | 2892 |
| ApproxLP (1) | 6.9e-04 | 2.7 | 2126 | 5740.2 |
| ApproxLP (2) | 4.3e-05 | 3.1 | 2890 | 8959 |
| OAM (0) | N/A | 1.9 | 1418 | 2694.2 |
| OAM (1) | 4.3e-04 | 2.3 | 2082 | 4788.6 |
| OAM (2) | 2.7e-05 | 2.7 | 2583 | 6974.1 |

## 4.3 Hybrid Approximation

There are a few approximate designs that combine the multipliers with different precisions together to adapt to the varying accuracy requirements [42,43], which are called hybrid approximation in this chapter. For example, reference [42] propose to combine accurate and approximate multipliers together to adjust the computational accuracy by selecting the appropriate multiplier. For the approximate multiplier, after detecting the number of consecutive 1's or 0's of the mantissa, the mantissa can then be rounded to 1 or 2, both of which make the multiplication as a shift operation. If a higher precision is required, the accurate multiplier is then invoked to conduct the calculation. Reference [43] uses 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 the consecutive bits of the inputs. However, such methods heavily rely on an accurate or high-precision multiplier, which significantly increases the circuit area. Furthermore, it is difficult to predict whether approximate or accurate computation should be conducted.

## 5 Approximate Multiplier with Circuit Level Approximation

This chapter discusses a few general circuit-level techniques for approximation, such as K-map modification, gate-level pruning, and voltage over-scaling (VOS), which are applicable to various architectures or algorithms.

### 5.1 K-Map Modification

Karnaugh map (K-map) is a common method for Boolean algebra expression simplification. The basic idea of K-map is to group the adjacent squares with the same logic values as much as possible. However, it is quite common in practice one or more squares cannot be grouped, causing additional logics and hence area. Thus, the approximation to K-map can be introduced to modify the adjacent square to the same value so as to group the squares and obtain a more compact representation. For example, the approximate multiplier UDM discussed in section 3 is comprised of a 2×2 multiplication module [17], which can be designed through K-map modification. By modifying the K-map as in Fig. 15, the basic block can act as both a partial product generator and a compressor with an error rate of 1/16 [17]. As shown in Fig. 16, when compared to the accurate logic implementation, the approximate implementation needs much fewer logic gates (37.5% reduction) with a shorter critical path.

The K-map modification can be applied to other arithmetic functions, such as adders [28], compressors [29, 30], and booth encoding modules [31, 32, 62]. For example, Yin *et al.* uses K-map modification to design an approximate modified Booth encoding (AMBE) module. With the modified K-map in Fig. 17, the original expression for modified Booth encoding algorithm:

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

can be simplified to [32]:

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

### 5.2 Gate-Level Pruning

Gate-level pruning provides an alternative to simplify netlist. It is based on the probabilistic pruning, which prunes less active gates from a circuit with limited accuracy loss [33]. Jeremy *et al.* propose to transform the circuit to a graph and prune the nodes with the lowest significance-activity product (SAP) during synthesis [34]. 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-

| $A_1A_0$ \ $B_1B_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 000 | 000 | 000 | 000 |
| 01 | 000 | 001 | 011 | 010 |
| 11 | 000 | 011 | $\boxed{111}$ | 110 |
| 10 | 000 | 010 | 110 | 100 |

**Fig. 15** An example of modifying K-map to achieve more compact design [17].

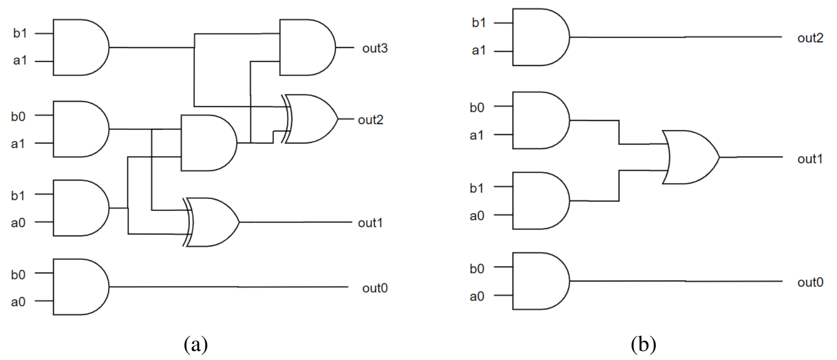


**Fig. 16** Comparison on the implementations of the $2 \times 2$ multiplier module: (a) Accurate logic implementation; (b) Approximate logic implementation [17].

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

**Fig. 17** A K-map example of AMBE [32].

defined and then backward propagated to calculate the significance of other gates. The activity of a node can be extracted from the .SAIF file (Switching Activity Interchange Format), which presents the toggle counts of wires. The digital design flow with gate-level pruning is presented in Fig. 18.

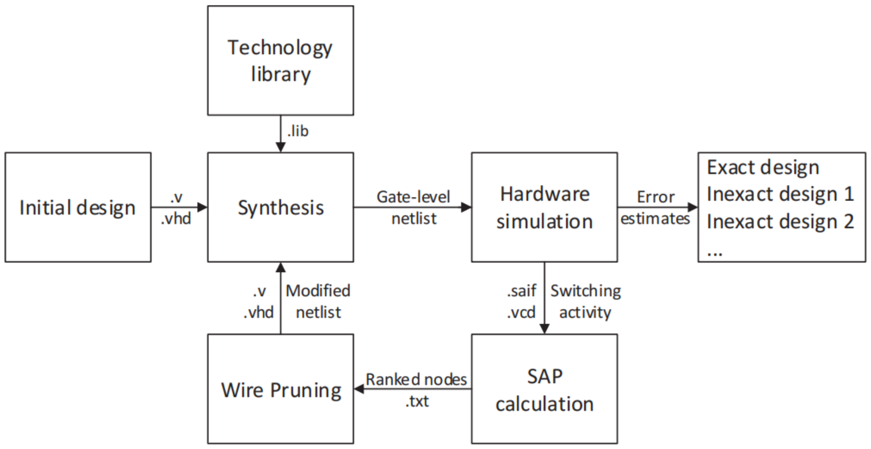


**Fig. 18** An example of gate-level pruning in digital design flow [34].

### 5.3 Voltage Over-scaling

Voltage scaling is a common method used to reduce power consumption [22]. In general, the operating supply voltage needs to be higher than $V_{dd-crit}$, which is the minimum supply voltage to ensure the timing of the critical path [23, 63, 64]. While voltage over-scaling (VOS) reduces power effectively, error is introduced inevitably. Hence, the key idea of VOS is to reduce the errors introduced by timing violations due to VOS [64]. 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 alleviate the impact of low supply voltage [23]. Liu *et al*. further propose an analytical method to assess computation errors due to VOS, which can then select the corresponding architecture and setup [23].

## 6 Conclusions

In this chapter, we reviewed approximate multipliers from three levels, *i.e.*, architecture, algorithm, and circuit. At architecture level, various approximation strategies were presented and discussed according to the stages of a conventional reference multiplier. At algorithm level, logarithm-based, linearization-based, and hybrid approximations were reviewed. Finally, at circuit level, we introduced three circuit level approximation techniques that can be applied together with any of the aforementioned approximation methods at architecture or algorithm levels. Detailed experimental results were presented in each section to help understand the pros and cons of different techniques at different design layers.

## References

1. L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.
2. V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "Impact: Imprecise adders for low-power approximate computing," in *IEEE/ACM International Symposium on Low Power Electronics and Design*, 2011, pp. 409–414.
3. J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS)*, 2013, pp. 1–6.
4. M. Imani, A. Rahimi, and T. S. Rosing, "Resistive configurable associative memory for approximate computing," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, ser. DATE '16. San Jose, CA, USA: EDA Consortium, 2016, p. 1327–1332.
5. S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46. New York, NY, USA: Association for Computing Machinery, 2013, p. 1–12. [Online]. Available: https://doi.org/10.1145/2540708.2540710
6. W. Liu, F. Lombardi, and M. Shulte, "A retrospective and prospective view of approximate computing [point of view," *Proceedings of the IEEE*, vol. 108, no. 3, pp. 394–399, 2020.
7. J. Deng, Z. Shi, and C. Zhuo, "Energy-efficient real-time uav object detection on embedded platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 3123–3127, 2019.
8. H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2108–2135, 2020.
9. Gokul Govindu, L. Zhuo, S. Choi, and V. Prasanna, "Analysis of high-performance floating-point arithmetic on fpgas," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, 2004, pp. 149–.
10. R. K. Yu and G. B. Zyner, "167 mhz radix-4 floating point multiplier," in *Proceedings of the 12th Symposium on Computer Arithmetic*, 1995, pp. 149–154.
11. M. Courbariaux, Y. Bengio, and J. David, "Low precision arithmetic for deep learning," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.7024
12. J. Deng and C. Zhuo, "Energy efficient real-time uav object detection on embedded platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, pp. 1–1, 12 2019.
13. K. He, A. Gerstlauer, and M. Orshansky, "Circuit-level timing-error acceptance for design of energy-efficient dct/idct-based systems," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 23, pp. 961–974, 06 2013.
14. M. Imani, Y. Kim, A. Rahimi, and T. Rosing, "Acam: Approximate computing based on adaptive associative memory with online learning," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ser. ISLPED '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 162–167. [Online]. Available: https://doi.org/10.1145/2934583.2934595
15. M. Imani, S. Patil, and T. 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*, ser. DATE '16. San Jose, CA, USA: EDA Consortium, 2016, p. 373–378.
16. M. Imani, M. Samragh, Y. Kim, S. Gupta, F. Koushanfar, and T. Rosing, "RAPIDNN: in-memory deep neural network acceleration framework," *CoRR*, vol. abs/1806.05794, 2018. [Online]. Available: http://arxiv.org/abs/1806.05794
17. P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *2011 24th Internatioal Conference on VLSI Design*. IEEE, 2011, pp. 346–351.

18. M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, "Invited: Cross-layer approximate computing: From logic to architectures," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.
19. A. Suhre, F. Keskin, T. Ersahin, R. Cetin-Atalay, R. Ansari, and A. E. Cetin, "A multiplication-free framework for signal processing and applications in biomedical image analysis," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 1123–1127.
20. S. Vahdat, M. Kamal, A. Afzali-Kusha, M. Pedram, and Z. Navabi, "Truncapp: A truncation-based approximate divider for energy efficient dsp applications," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 1635–1638.
21. C. Zhuo, K. Unda, Y. Shi, and W.-K. Shih, "From layout to system: Early stage power delivery and architecture co-exploration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, pp. 1–1, 05 2018.
22. A. P. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital cmos circuits," *Proceedings of the IEEE*, vol. 83, no. 4, pp. 498–523, 1995.
23. Y. Liu, T. Zhang, and K. K. Parhi, "Computation error analysis in digital signal processing systems with overscaled supply voltage," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 18, no. 4, pp. 517–526, 2009.
24. M. Imani, A. Sokolova, R. Garcia, A. Huang, F. Wu, B. Aksanli, and T. Rosing, "Approxlp: Approximate multiplication with linearization and iterative error control," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.
25. C. Chen, S. Yang, W. Qian, M. Imani, X. Yin, and C. Zhuo, "Optimally approximated and unbiased floating-point multiplier with runtime configurability," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.
26. "Ieee standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.
27. C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on electronic Computers*, no. 1, pp. 14–17, 1964.
28. 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)*. IEEE, 2018, pp. 410–415.
29. N. Van Toan and J.-G. Lee, "Fpga-based multi-level approximate multipliers for high-performance error-resilient applications," *IEEE Access*, vol. 8, pp. 25 481–25 497, 2020.
30. M. Ha and S. Lee, "Multipliers with approximate 4–2 compressors and error recovery modules," *IEEE Embedded Systems Letters*, vol. 10, no. 1, pp. 6–9, 2017.
31. L. Qian, C. Wang, W. Liu, F. Lombardi, and J. Han, "Design and evaluation of an approximate wallace-booth multiplier," in *2016 IEEE international symposium on circuits and systems (ISCAS)*. IEEE, 2016, pp. 1974–1977.
32. P. Yin, C. Wang, W. Liu, E. E. Swartzlander, and F. Lombardi, "Designs of approximate floating-point multipliers with variable accuracy for error-tolerant applications," *Journal of Signal Processing Systems*, vol. 90, no. 4, pp. 641–654, 2018.
33. A. Lingamneni, C. Enz, J.-L. Nagel, K. Palem, and C. Piguet, "Energy parsimonious circuit design through probabilistic pruning," in *2011 Design, Automation & Test in Europe*. IEEE, 2011, pp. 1–6.
34. J. Schlachter, V. Camus, C. Enz, and K. V. Palem, "Automatic generation of inexact digital circuits by gate-level pruning," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2015, pp. 173–176.
35. V. Camus, J. Schlachter, C. Enz, M. Gautschi, and F. K. Gurkaynak, "Approximate 32-bit floating-point unit design with 53% power-area product reduction," in *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference*, 2016, pp. 465–468.
36. S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim, "Energy-efficient approximate multiplication for digital signal processing and classification applications," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 23, no. 6, pp. 1180–1184, 2014.

37. K. Bhardwaj, P. S. Mane, and J. Henkel, “Power- and area-efficient approximate wallace tree multiplier for error-resilient systems,” in *Fifteenth International Symposium on Quality Electronic Design*, 2014, pp. 263–269.
38. C. Liu, J. Han, and F. Lombardi, “A low-power, high-performance approximate multiplier with configurable partial error recovery,” in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–4.
39. C. Lin and I. Lin, “High accuracy approximate multiplier with error correction,” in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, 2013, pp. 33–38.
40. C. Guo, L. Zhang, X. Zhou, W. Qian, and C. Zhuo, “A reconfigurable approximate multiplier for quantized cnn applications,” in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020, pp. 235–240.
41. S. E. Ahmed, S. Kadam, and M. Srinivas, “An iterative logarithmic multiplier with improved precision,” in *2016 IEEE 23nd Symposium on Computer Arithmetic (ARITH)*. IEEE, 2016, pp. 104–111.
42. M. Imani, D. Peroni, and T. Rosing, “Cfpu: Configurable floating point multiplier for energy-efficient computing,” in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2017, pp. 1–6.
43. M. Imani, R. Garcia, S. Gupta, and T. Rosing, “Rmac: Runtime configurable floating point multiplier for approximate computing,” in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2018, pp. 1–6.
44. E. de Angel and E. Swartzlander, “Low power parallel multipliers,” in *VLSI Signal Processing, Ix*. IEEE, 1996, pp. 199–208.
45. S. Hashemi, R. I. Bahar, and S. Reda, “Drum: A dynamic range unbiased multiplier for approximate applications,” in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2015, pp. 418–425.
46. S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram, “Tosam: An energy-efficient truncation-and rounding-based scalable approximate multiplier,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 5, pp. 1161–1173, 2019.
47. ——, “Letam: A low energy truncation-based approximate multiplier,” *Computers & Electrical Engineering*, vol. 63, pp. 1–17, 2017.
48. S. Venkatachalam and S.-B. Ko, “Design of power and area efficient approximate multipliers,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1782–1786, 2017.
49. T. Yang, T. Ukezono, and T. Sato, “A low-power high-speed accuracy-controllable approximate multiplier design,” in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2018, pp. 605–610.
50. ——, “Low-power and high-speed approximate multiplier design with a tree compressor,” in *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 2017, pp. 89–96.
51. A. Momeni, J. Han, P. Montuschi, and F. Lombardi, “Design and analysis of approximate compressors for multiplication,” *IEEE Transactions on Computers*, vol. 64, no. 4, pp. 984–994, 2014.
52. C.-W. Tung and S.-H. Huang, “Low-power high-accuracy approximate multiplier using approximate high-order compressors,” in *2019 2nd International Conference on Communication Engineering and Technology (ICCET)*. IEEE, 2019, pp. 163–167.
53. Z. Yang, J. Han, and F. Lombardi, “Approximate compressors for error-resilient multiplier design,” in *2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*. IEEE, 2015, pp. 183–186.
54. H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, “Bio-inspired imprecise computational blocks for efficient vlsi implementation of soft-computing applications,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 4, pp. 850–862, 2009.
55. H.-L. Lin, R. C. Chang, and M.-T. Chan, “Design of a novel radix-4 booth multiplier,” in *The 2004 IEEE Asia-Pacific Conference on Circuits and Systems*, vol. 2. Citeseer, 2004, pp. 837–840.

56. H. Jiang, J. Han, F. Qiao, and F. Lombardi, “Approximate radix-8 booth multipliers for low-power and high-performance operation,” *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2638–2644, 2015.
57. J. N. Mitchell, “Computer multiplication and division using binary logarithms,” *IRE Transactions on Electronic Computers*, no. 4, pp. 512–517, 1962.
58. W. Liu, J. Xu, D. Wang, C. Wang, P. Montuschi, and F. Lombardi, “Design and evaluation of approximate logarithmic multipliers for low power error-tolerant applications,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 9, pp. 2856–2868, 2018.
59. H. Saadat, H. Bokhari, and S. Parameswaran, “Minimally biased multipliers for approximate integer and floating-point multiplication,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2623–2635, 2018.
60. M. S. Ansari, B. F. Cockburn, and J. Han, “A hardware-efficient logarithmic multiplier with improved accuracy,” in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 928–931.
61. ——, “An improved logarithmic multiplier for energy-efficient neural computing,” *IEEE Transactions on Computers*, vol. 70, no. 4, pp. 614–625, 2020.
62. W. Liu, L. Qian, C. Wang, H. Jiang, J. Han, and F. Lombardi, “Design of approximate radix-4 booth multipliers for error-tolerant computing,” *IEEE Transactions on Computers*, vol. 66, no. 8, pp. 1435–1441, 2017.
63. D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, “Design of voltage-scalable meta-functions for approximate computing,” in *2011 Design, Automation & Test in Europe*. IEEE, 2011, pp. 1–6.
64. J. Chen and J. Hu, “Energy-efficient digital signal processing via voltage-overscaling-based residue number system,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 7, pp. 1322–1332, 2012.