Que HH, Jin Y, Wang T *et al.* A survey of approximate computing: From arithmetic units design to high-level applications. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 38(2): 251-272 Mar. 2023. DOI: 10.1007/s11390-023-2537-y

A Survey of Approximate Computing: From Arithmetic Units Design to High-Level Applications

Hao-Hua Que[1] (阙浩华), Yu Jin[1] (金　雨), Tong Wang[1] (王　童), Ming-Kai Liu[1] (刘明楷)
Xing-Hua Yang[1, *] (杨兴华), and Fei Qiao[2] (乔　飞), *Senior Member, CCF, Member, IEEE*

[1] *College of Science, Beijing Forestry University, Beijing 100091, China*

[2] *Department of Electronic Engineering, Tsinghua University, Beijing 100084, China*

E-mail: qh13005968844@bjfu.edu.cn; AUH2OJin@bjfu.edu.cn; WangTom@bjfu.edu.cn; KleinKai565@bjfu.edu.cn
yangxh@bjfu.edu.cn; qiaofei@tsinghua.edu.cn

Received May 28, 2022; accepted March 16, 2023.

Abstract Realizing a high-performance and energy-efficient circuit system is one of the critical tasks for circuit designers. Conventional researchers always concentrated on the tradeoffs between the energy and the performance in circuit and system design based on accurate computing. However, as video/image processing and machine learning algorithms are widespread, the technique of approximate computing in these applications has become a hot topic. The errors caused by approximate computing could be tolerated by these applications with specific processing or algorithms, and large improvements in performance or power savings could be achieved with some acceptable loss in final output quality. This paper presents a survey of approximate computing from arithmetic units design to high-level applications, in which we try to give researchers a comprehensive and insightful understanding of approximate computing. We believe that approximate computing will play an important role in the circuit and system design in the future, especially with the rapid development of artificial intelligence algorithms and their related applications.

Keywords approximate computing, arithmetic unit, low power, high performance, reduced output quality

1 Introduction

The concept of approximation is well-established in many fields, such as physics (e.g., saddle point approximation, quantum mechanical perturbation solving), engineering (e.g., linear approximation), and medicine (e.g., medical image noise reduction). However, approximations have been introduced into different fields for various reasons. Despite the continuous advances in semiconductor technology and energy-efficient design techniques, the computation demands of large-scale modern systems such as scientific computing, data analysis, and financial transactions have increased significantly in recent years, especially as computer systems become increasingly mobile and embedded with various sensors to interact with the physical world. The overall energy consumption of these systems continues to grow exponentially at an alarming rate. As a result, approximations have been introduced to address this growing power and performance challenges[1].

Approximate computing could achieve large improvements in performance and energy efficiency by relaxing the requirement of output quality for the system, exploiting the gap between the level of accuracy required by the user and the level of accuracy provided by the computing system to obtain various optimizations. For example, in image processing applica-

Survey

Special Section on Approximate Computing Circuits and Systems

This work was supported by the Fundamental Research Funds for the Central Universities of China under Grant No. BLX202015, Beijing Municipal Natural Science Foundation under Grant No. 6222038, and the National Natural Science Foundation of China under Grant No. 92164203.

*Corresponding Author

tions where the output quality of the computing system is evaluated through the human perspective, the statement of "good image quality" is very subjective, and each person has different image analysis capabilities. With this regard, approximate image processing can be implemented with tolerable computation errors, which implies some loss of output quality.

Approximate computing could be implemented through a variety of strategies, from the hardware level to various layers of software applications. The deployment of approximate computing at different levels has a different impact on final output quality. At the hardware level of the integrated circuit design in nano-scale era, the possibility of exhibiting uncertainties and errors for a CMOS device is increasing substantially, which will lead to additional power consumption as redundant components to be used to improve the reliability of the circuit. Applying approximation techniques to these integrated circuit designs could reduce the additional power consumption by introducing some acceptable errors. Besides, various storage and processing architectures have been proposed to support the implementation of approximate computing units with analytical methods and design metrics, such as approximate adders, multipliers and dividers. At the application level, machine learning can also utilize approximate computing techniques. The training process of machine learning can tolerate some accuracy loss, as in Google's deep learning chip, the tensor processing units use accuracy scaling, resulting in significant performance improvement. The introduction of approximate computing in deep neural networks (DNNs) includes quantization and weight pruning, which could make DNNs easier to deploy and less computationally intensive with reduced memory usage and computation complexity.

The introduction of approximation into a computing system has some intrinsic effects. The benefits of approximate computing have already been mentioned above; however, approximate computing also has its limitations, as it will reduce the intrinsic reliability and accuracy of the results. In some applications where very high computation accuracy is required (e.g., aerospace systems), the approximate computing may lead to unexplained system crashes. Obviously, approximate computing is not suitable in these applications. It should also be noted that even when approximate computing is used in the field of image processing or machine learning, the output quality should be evaluated completely and thoroughly.

In fact, there have been many review papers on approximate computing up to now. Approximation techniques, in particular approximate multipliers, and generalized approximate high-level synthesis approaches are introduced in detail in [2]. Machine learning and neutral networks are used in addition to these approximate computing technologies in [2], in order to provide some corresponding improvements such as in circuit performance. In [3], basic arithmetic units such as approximate adders, multipliers and dividers with multiple circuit structures are reviewed. In [4], in addition to the introduction of the approximate logic units, the circuit structure of approximate memory blocks is described in detail. Compared with all of these surveys, this paper presents a survey of approximate computing from the arithmetic units design to high-level applications, in which we try to give researchers a comprehensive and insightful understanding of approximate computing. As shown in Fig.1, the structure of this survey is as follows. Section 2 introduces approximate computing from the arithmetic units design to practical applications, which is the main content of this survey. Be-

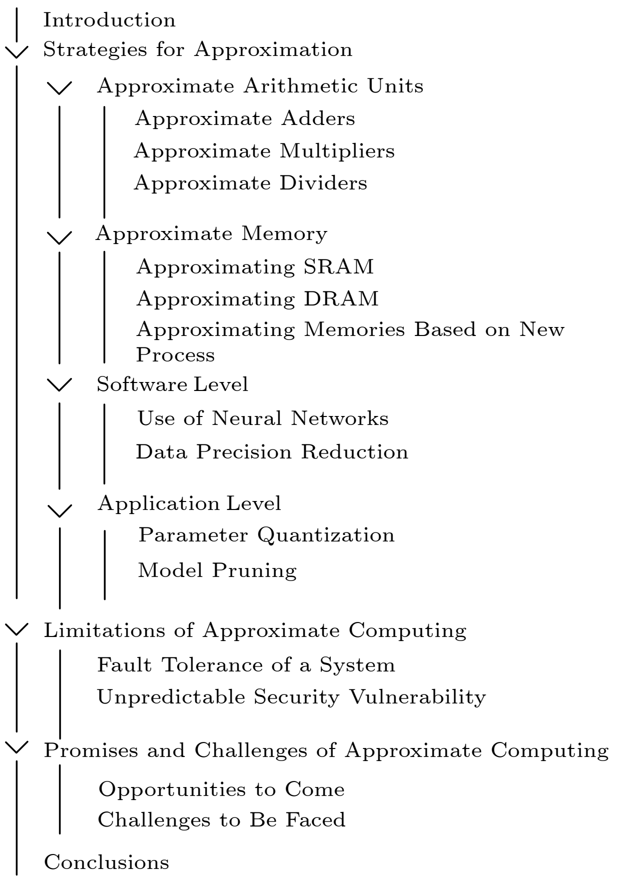

Fig.1. Organization of the paper.

sides, Section 3 describes the limitations for approximate computing. Section 4 describes the promises and challenges of approximate computing. Conclusions are presented in Section 5.

2 Strategies for Approximate Computing

The researches and applications of approximation techniques cover the entire circuit and system levels, including approximating feld-programmable gate arrays (FPGAs), embedded systems, general computers, and graphics processing units (GPUs). We divide the techniques of approximate computing into four groups and defines their implementation levels: approximate arithmetic units, approximate memory, and approximate computing in software and application levels. All of these approximate computing techniques are described in detail as follows.

2.1 Approximate Arithmetic Units

Adders, multipliers and dividers are indispensable computational units in circuit systems. Therefore, researchers have carried out tremendous work on the computational design, from the transistor level to the gate circuit level, and even to specific computational methods for addition, multiplication and division.

The approximate computation at the CMOS device level was first proposed by Palem[5], in order to construct the basic approximate computational unit, namely PCMOS (Probabilistic CMOS). According to the relevant exposition in the work, the lower bound on the energy required for a single flip-flop of a single-bit inverter in a perfectly correct output condition is $(\ln 2) \times kT$ joules (k is the Boltzmann constant, and T is the thermodynamic temperature scale). If the probability of the correct output is no longer 1, denoted as p $(0.5 < p < 1)$, the lower limit of energy required for one flip is $(\ln 2p) \times kT$. As shown in Fig.2, since the probability of the correct flip and the energy consumed show an exponential relationship, a small fraction of flip errors will lead to significant energy saving at the device level. Thus, the basic approximate computing unit can be constructed with this theoretical basis. As the flip errors are introduced mainly by reducing the supply voltage, in which the supply voltage is close to the threshold voltage of the CMOS device, the device's speed will drop significantly at this time. Supposing the operating frequency is not reduced accordingly, the circuit

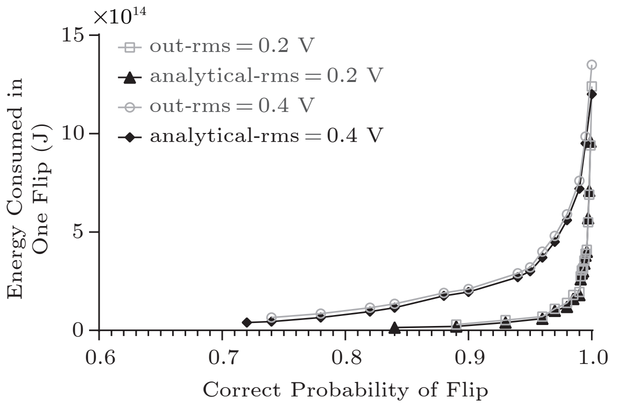

Fig.2. Relationship between flip probability and energy consumption for PCMOS[5]. out-rms: output-root means square.

with lower supply voltage will be further subject to timing sampling errors on top of logic computing, which are much larger than the flip errors of the CMOS device near the threshold voltage. The reasons as mentioned above make PCOMS, although being more promising at the theoretical level, encounter big obstacles in practical applications.

2.1.1 *Approximate Adders*

Adders have a vital position in circuit systems, and the design of adders with low power consumption and high-performance characteristics has been long-standing in the traditional circuit design. In the field of approximate computing, the design of approximate adders has also received much attention.

Gupta *et al.*[6] proposed a deletion method for single-bit full adders at the transistor level, as shown in Fig.3. The basic idea is to partially remove the original transistor structure of an accurate 1-bit mirror adder (shown in Fig.3(a)) to obtain a 1-bit approximate adder, as shown in Fig.3(b). The deleted transistors reduce the overall circuit capacitance and help to reduce the circuit power consumption and delay, improving the circuit energy efficiency. At the same time, removing the transistor causes the 1-bit mirror adder to produce a partial error output. In fact, the authors designed 1-bit approximate full adders with different degrees of approximation. Reducing the number of transistors results in increased energy efficiency, but also leads to higher levels of approximation and output error. Finally, the original 1-bit full adder and the approximate full adder are combined. For example, in Ripple Carry Adder (RCA), the exact full adders are used in the high bits, and the approximate full adders are used in the low bits. In this

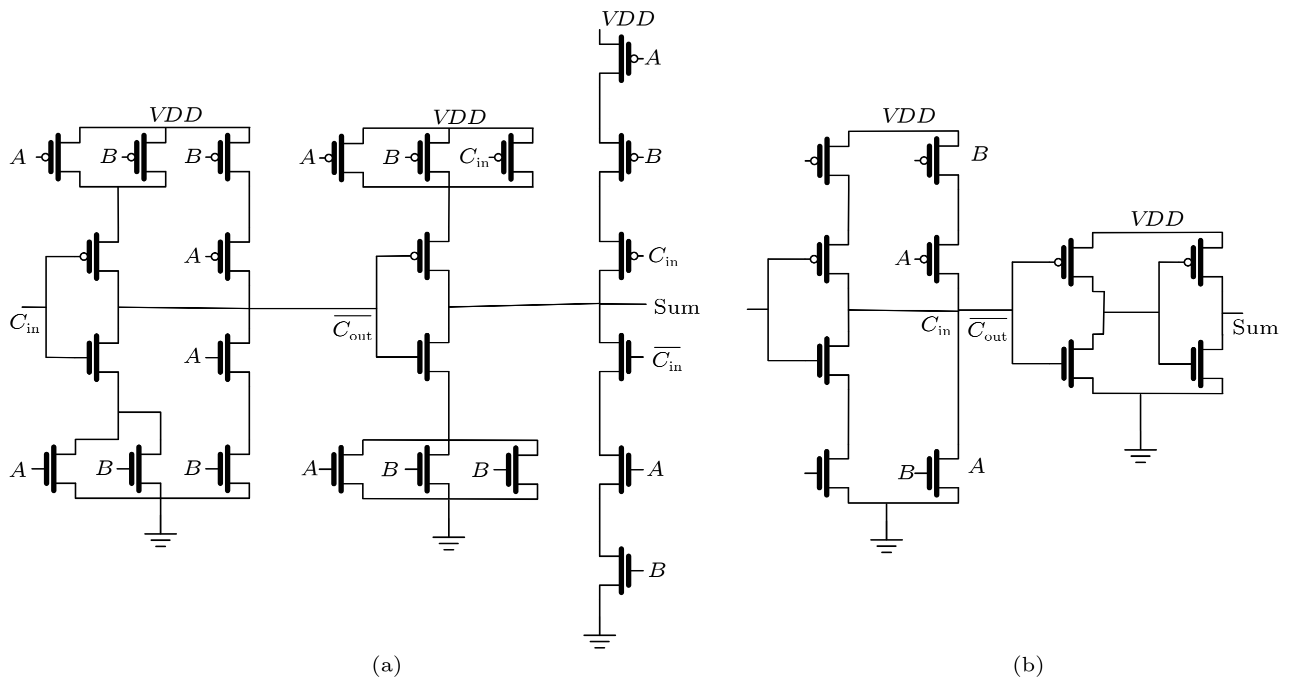
VDD
A
B
B
A
C_{in}
A
A
B
B
C_{in}
$\overline{C_{\text{out}}}$
Sum
$\overline{C_{\text{in}}}$
(a)
B
C_{in}
$\overline{C_{\text{out}}}$
Sum
(b)

Fig.3. Transistor level design[6]. (a) 1-bit accurate adder. (b) 1-bit approximate adder.

way, the entire RCA is incorrectly computed in the low-bit output, and its power consumption and latency will be effectively reduced. This design is verified in Discrete Cosine Transform (DCT) in image processing, which shows superior output quality results to an adder structure with direct truncation. However, the designer has to reconstruct the circuit layout in practice because the original circuit of the full adder needs to be deleted at the transistor level. This process makes the application of this technique extremely inconvenient, especially in the design of circuit systems with different computational bit widths and different accuracy requirements, where the workload is high and the design efficiency is low.

Given the large cost required to design approximate adders at the transistor level, researchers have performed many designs at the gate circuit level[7–10]. These designs have approximated the conventional RCA or carry-lookahead adder (CLA), in which the main idea is to shorten the critical path by approximation methods. However, different approximation structures can lead to different output errors and energy efficiency gains. It is important to note that even if an approximate adder has very low power consumption or delay, it is still unusable in real applications if its output error is too large.

As shown in Fig.4, Zhu *et al.*[8] attempted to segment the critical path of the conventional RCA by prediction, such as dividing the N-bit RCA into several groups with M-bit for one group ($M < N$), and the whole RCA will be divided into N/M groups, $a_{n-1:0}$ and $b_{n-1:0}$ are the input data, and $s_{n-1:0}$ is the output data. The carry signal for each group is predicted by corresponding circuits, in which only part of the input data will be used. As we know, in conventional CLA, the circuits to predict the carry signal are very complex since all the input data should be used so that the final result is correct. In this design, the input for the prediction of the carry signal is only part of the input data, and then the sum signal of each group may be subject to error. By the above method, the critical path of the whole adder is significantly reduced. Theoretically, there is only one stage of carry signal prediction delay and the delay of the M-bits summing block. Reducing the critical path delay can effectively improve the computational circuit's operation speed or reduce the circuit power consumption by lowering the supply voltage while keeping the speed constant. Both cases can essentially reduce the circuit operation energy. This gate-level approximate design offers a high degree of flexibility and a simple design process that can be directly synthesized by Design-Compiler tools using a standard circuit description language.

However, unfortunately, the output error of this approximate adder is significant; although it is highly energy-efficient in operation, in practice, the large output error makes the final output of the circuit sys-

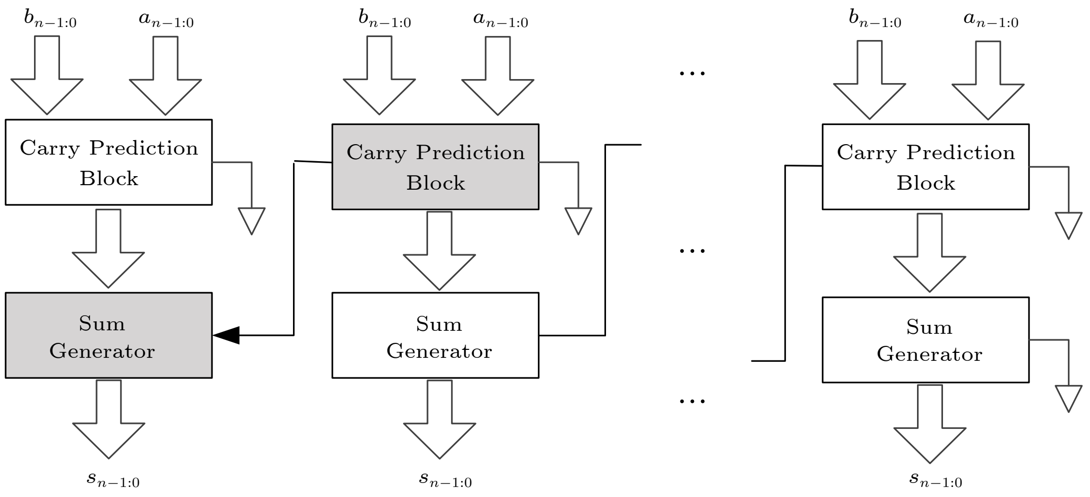

Fig.4. Approximate adder with partial prediction[8].

tem using this approximate unit unable to meet the output quality requirements. The main reason for the large output error is that an incorrect prediction to the high carry signal will cause the final result to be incorrect in the high output bits. Thus, Kim *et al.*[7] modified the carry signal predictor and tried to compensate for the output error, in which the RCA is also divided into several groups, and each group consists of two parts, one is the carry signal prediction block, and the other is implemented with an RCA scheme that calculates the current sum output. A multiplexer (MUX) is inserted between two adjacent groups. It should be noted that the carry signal of group $(i+1)$ is connected to the carry signal prediction blocks (i) and $(i-1)$ through the MUX. For each input data, if each bit of the input data in part (i) is different, the output of carry signal prediction block $(i-1)$ will be used. Otherwise, the value of block (i) will be taken. For the final addition result, if each bit of the input data in parts (i) and $(i-1)$ is different, and the predicted carry signal for part $(i+1)$ value is detected to be different from the real carry signal, then the approximate result will be forcibly set to "1" in parts (i) and $(i-1)$ as an error compensation. By the above measures, the approximate adder reduces the mean square value of the error by three orders of magnitude compared with the approximate adder in [8]. However, in practical applications, this magnitude of output errors is still too large to make the adder used in specific applications.

This method of segmenting the critical path of the adder into different groups is also applied in several others designs, such as Accuracy-Configifurable Adder (ACA)[8], and approximate adder with correct sign calculation[10]. Among all of these researches, it is worth noting that the approximate adder proposed in [10] has a unique advantage, since a correct sign calculation for 2's complement signed additions is ensured. This improvement is very meaningful, as in real computation tasks, the input data is always in 2's complement formation. The output error may be quite large if the final sum result has a wrong sign, which will further crash the entire computing system.

The approximate adders mentioned above use the conventional CMOS process to construct the whole cell and have a common feature that regardless of the approximation method, these adders will complete the computation in one clock cycle. In addition to these adders, there are also variable latency approximate adders[11], which will complete the computation within various clock cycles. In some other researches[12], approximate adders are implemented based on novel process, such as quantum-dot cellular automata (QCA)[13].

Fig.5 shows an approximate adder with the variable latency scheme as proposed in [11]. In essence, this kind of approximate adders also try to cut the critical path of each adder into several short paths. After inserting a series of prediction blocks, there is a risk of prediction errors at each prediction block, which can lead to an error in the output. A variable latency scheme is used mainly because of the special requirement that the high bits of the output must be accurately computed. In the single-cycle approximate adder described above, no matter how the designers use error compensation, errors will be inevitably introduced in the high bits, even though the probability of such errors occurring may not be high. In contrast, the designer performs exact computation for the high bits with variable clock cycles in the approxi-

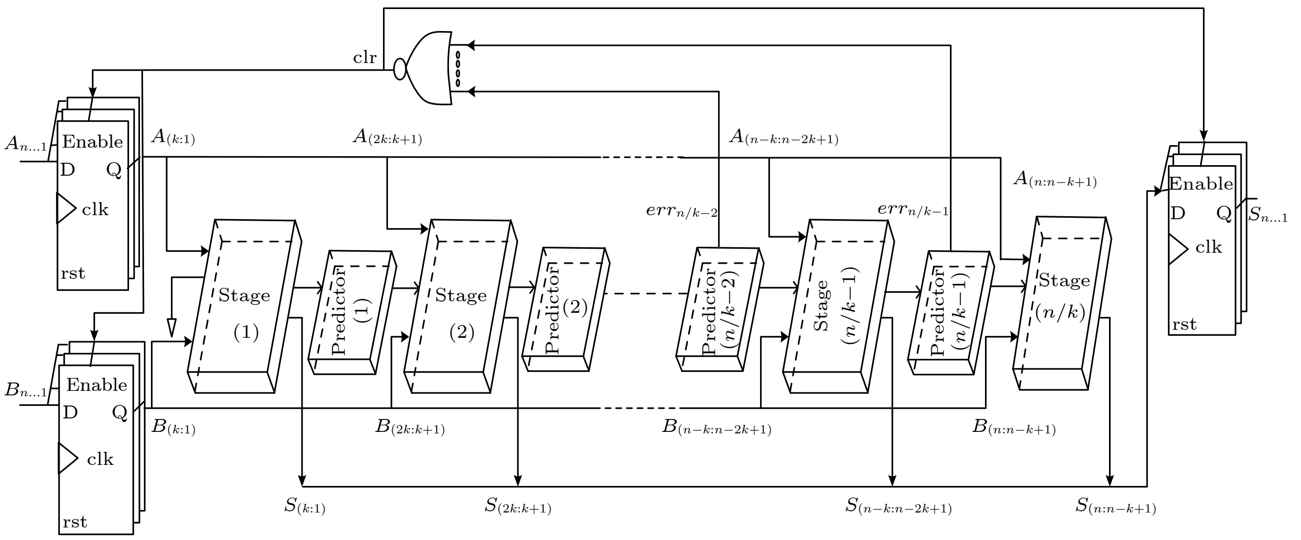

Fig.5. Approximate adder with variable latency scheme[11]. clr: clear; clk: clock; rst: reset; D: input of D-flip-flop; Q: output of D-flip-flop.

mate adder. For the low bits, approximate computation will be used without extra cycles. As shown in Fig.5, taking a 32-bit adder as an example ($A_{n\dots1}$ and $B_{n\dots1}$ are the input data; $S_{(k:1)}$, $S_{(2k:k+1)}$, $\cdots$, $S_{(n-k:n-2k+1)}$ and $S_{(n:n-k+1)}$ are the output data; $err_{n/k-2}$ and $err_{n/k-1}$ are the signal predicting where is a wrong speculation), every four bits are divided into a group, and seven prediction blocks need to be inserted. For the first three prediction blocks, if the predicted carry signals do not match the real ones, the prediction blocks will generate error signals. These error signals will latch the input and output flip-flops, and the real carry signals will be passed forward. After several clock cycles, the first 12 bits of the sum-output must be calculated correctly. As for the last four prediction blocks, no error signal is issued even if an error occurs in prediction blocks, which means that no clock cycles will be consumed.

It can be seen that there are two benefits using a variable latency scheme. One is that it can effectively be ensured that the high bits do not introduce any errors. The other is that the output errors and performance improvements can be flexibly configured over a wide range. However, this structure also has enormous drawbacks. The number of clocks required for each addition operation can be a random number, leading to a relatively significant burden on the entire pipeline when designing synchronous timing logic circuits.

In addition to using the traditional CMOS process and the Boolean logic gate, new process technologies have been applied in approximate adder design[12]. It should be noted that these new process technologies rely on majority logic (ML), which is a different framework from conventional Boolean logic. As shown in Fig.6(a), the inputs are A, B, C and the output is F. The logic expression of this ML gate is $F = M(A, B, C) = AB + BC + AC$. Thus, for one bit

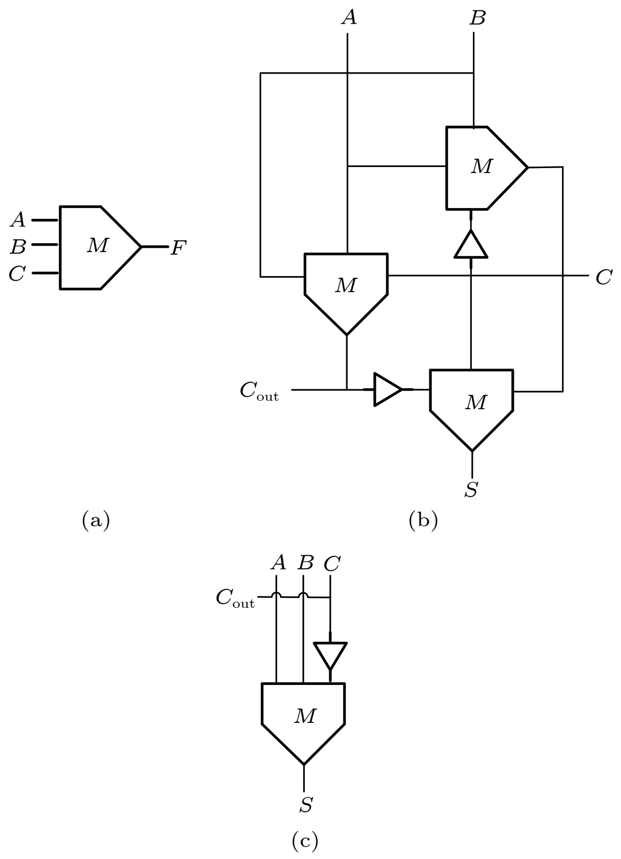

Fig.6. (a) Majority logic gate[12]. (b) Accurate adder with ML[12]. (c) Approximate adder with ML[12].

accurate adder, the final sum (S) and carry (C) signals can be expressed with ML as:

$$C_{\text{out}} = AB + BC + AC = M(A, B, C),$$
$$S = A \oplus B \oplus C = M(\overline{C_{\text{out}}}, M(A, B, C), C).$$

The corresponding scheme is shown in Fig.6(b). With this basic circuit scheme, Zhang *et al.*[12] proposed a one-bit approximate adder based on ML. The authors observed that C_{out} and C are almost same except for two cases. Thus, the proposed approximate adder is expressed as:

$$C_{\text{out}} = C,$$
$$S = M(\overline{C_{\text{out}}}, M(A, B, \overline{C}), C) = (A, B, \overline{C}).$$

The corresponding scheme is shown in Fig.6(c). It can be seen that two ML gates can be saved, which means that the energy efficiency of the adder will be improved with certain output error. In essence, this approach of simplifying the truth table can also be used to approximate the adder design with CMOS process and Boolean logic. In [12], the authors evaluated the proposed ML-based approximate adder with QCA technology, which shows large improvements in performance and energy efficiency due to the approximate design method.

2.1.2 Approximate Multipliers

Compared with adders, multipliers are more complex and consume relatively more energy and delay. At the same time, in large-scale machine learning tasks, many convolution operations are computed by multiplication-accumulation, and thus the design of low-power, high-performance approximate multipliers has received extensive attention in past researches. Generally, a multiplier consists of three stages: partial product generation, partial product accumulation and final addition. Among all these stages, the partial product accumulation consumes the most delay and power. In essence, whether it is an accurate multiplier or an approximate multiplier, all the designers try to optimize the process of partial product accumulation to reduce its delay and power consumption. In accurate multiplier design, Wallace tree, Dadda tree and carry-save adder array[14] have been proposed to improve the speed of the multiplier. For the Wallace tree multiplier, no carry propagation is generated as the accumulation for every three partial products is operated in parallel. In fact, these accumulations can be implemented with (3: 2) compressors or (4: 2) compressors.

In past researches, there are different kinds of approximate multipliers: 1) approximate recursive multiplier using inaccurate 2 $\times$ 2 block to generate approximate partial products[15–17]; 2) multiplies applying approximation in partial product tree, including truncation, approximate accumulation using inaccurate adders or compressors[18–21]; 3) approximate logarithmic multipliers and booth multipliers[22–26].

For approximate recursive multipliers, the very first to use the approximate 2 $\times$ 2 multiplier to generate partial products is proposed by Kulkarni *et al.*[15]. Considering A_1A_0 and B_1B_0 as the input for the 2 $\times$ 2 multiplier, they approximated the truth table of the 2-bit multiplier as shown in Table 1. All the values in the truth table are correct except one, i.e., A_1A_0 and B_1B_0, where the accurate result should be "1001" but incorrectly replaced by "111". Although this step of approximation is small, the final implementation of the digital circuit saves a large number of logic gates, and thus the speed and energy efficiency of the whole circuit is greatly improved, as shown in Fig.7(a) and Fig.7(b). It can be seen that XOR-gate is no longer needed in approximate implementation and the whole critical path is reduced substantially. Thus, the partial products of a longer multiplier could be generated with this approximate 2 $\times$ 2 multiplier, as shown in Fig.7(c). For the 4 $\times$ 4 multiplier, A_H and X_H are the upper two bits, and A_L and X_L are the lower two bits. Using the proposed 2 $\times$ 2 multiplier, the partial products could be generated, and then approximation will be introduced through the partial products and the remaining accumulations are accurately computed.

Table 1. Truth Table for Approximate 2 $\times$ 2 Multiplier[15]

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

Some similar work is also shown in [16]; however, Waris *et al.*[17] pointed out the potential problems of the above approximation multiplier, as the errors generated by previous approximate multipliers are unidirectional. Thus, the errors of partial products from this approximate building block will be accumulated in one direction, leading to a large output error at last. Waris *et al.*[17] argued that the approximate multiplier has been simplified in the circuit in [15], but there is still more space for exploration. With all of

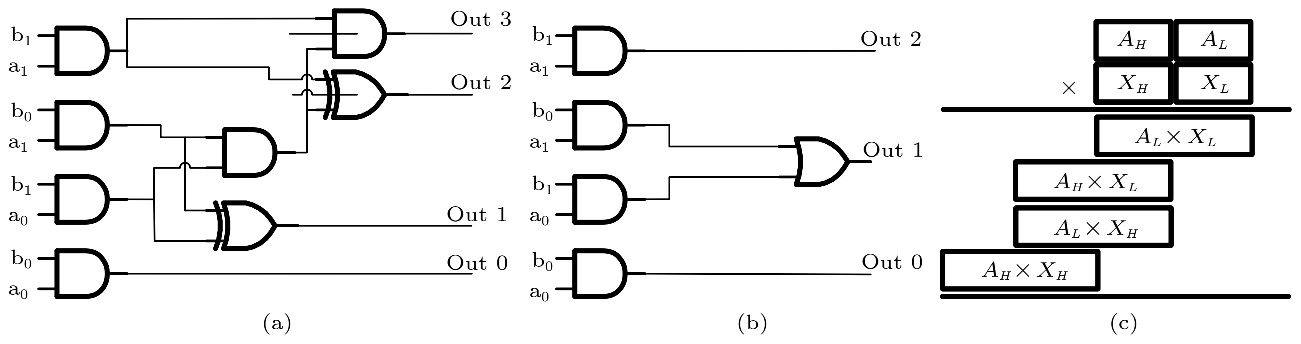

Fig.7. (a) Accurate 2 × 2 multiplier. (b) Approximate 2 × 2 multiplier. (c) Using the 2 × 2 multiplier as a building block to generate longer bits multiplier.

these observations, Waris *et al.*[17] proposed two kinds of improved 2 × 2 approximate adders, and the truth tables are shown in Table 2 (called $Mul2_a$ and $Mul2_b$). In $Mul2_a$, five outputs are approximated and in $Mul2_b$, six inaccurate outputs are generated. It should be noted that in both $Mul2_a$ and $Mul2_b$, the generated error has both positive and negative values. This is a big advantage when using $Mul2_a$ or $Mul2_b$ to generate partial products of a longer-bit multiplier, as the error for the final accumulation of partial products will be compensated, which will lead to a much smaller output error for the approximate multiplier.

In [18, 19], the designers turn attention to the approximation of partial products accumulation, which is different from the design in [15–17], where the partial products are generated with approximation errors, but accumulated using exact operations. However, in [18, 19], truncation and approximate compressors are utilized to accomplish the accumulation processing. In [18], an approximate multiplier is proposed by omitting some carry-save adders (CSA) as shown in Fig.8. In essence, the most straightforward way is to truncate some least significant bits (LSBs) of the input data, which means that no partial products and corresponding accumulations for LSBs are needed. But, the truncation method will cause too large output error for practical applications. Another approach for approximate partial products accumulation is to use inaccurate (4: 2) compressors, which are proposed in [19]. It is worth noting that the design method of these approximate compressors uses the same idea of simplifying the truth table. Depending on the degree of approximation, the designers try to compromise between the output error and circuit performance. As a general design principle, all the designers want the output error of the approximate multiplier to have a

Table 2. Truth Table of Improved Approximate 2 × 2 Multiplier[17]

Input				Exact Truth Table				$Mul2_a$ Truth Table				$Mul2_b$ Truth Table			
a_1	a_0	b_1	b_0	C_3	C_2	C_1	C_0	C_3	C_2	C_1	C_0	C_3	C_2	C_1	C_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	1×	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1
0	1	1	0	0	0	1	0	0	0	1	0	0	0	0×	0
0	1	1	1	0	0	1	1	0	0	1	1	0	0	0×	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1×	0
1	0	0	1	0	0	1	0	0	0	1	0	0	1	0	1×
1	0	1	0	0	1	0	0	0	1	0	0	0	1	1	0
1	0	1	1	0	1	1	0	0	1	1	1×	0	1	1	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1×	0
1	1	0	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1	0×	1×	1×	1	0×	1×	1×	1

Note: ×: not correct.

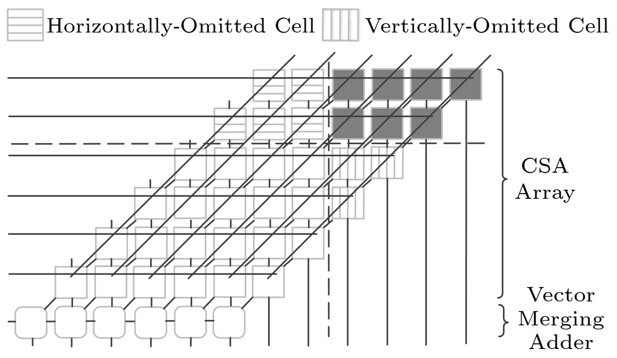

Fig.8. Approximate accumulation array in multiplier design[18].

standard or uniform distribution with zero mean value.

In [20], an interesting approximate method to multiplier is proposed, which is different from all the above work[15–19]. The main idea is that each of the input data for multiplication is approximated to the power n (2^n). Denoting the approximate input data of A and B by A_r and B_r respectively, then the final multiplication can be expressed as:

$$A \times B = (A_r - A) \times (B_r - B) + A_r \times B + B_r \times A - A_r \times B_r.$$

The most important observation is that, if we eliminate $(A_r - A) \times (B_r - B)$, which is hard to be computed, the multiplications of $A_r \times B_r$, $A_r \times B$, and $B_r \times A$ can be implemented just by the shifter units. The final result could be obtained by adding these three shifting outputs. This means the approximation of $(A_r - A) \times (B_r - B)$ will be introduced, but the whole implementation complexity of a multiplier is reduced substantially. The designers in [20] compared the proposed multiplier with some other designs in [21], and showed various degrees in accuracy and energy efficiency improvement.

In addition to applying approximation in partial product generation and accumulation, researchers also have applied approximation in booth multipliers[22–24], logarithmic multipliers[25, 26], and majority-logic (ML) based multipliers using new materials[27]. It is well known that in booth multipliers, after encoding the input data, the partial products could be reduced so that large improvement in performance and energy efficiency could be achieved. Approximate radix-4 booth multipliers are first proposed in [22, 23] in which the partial products are approximated as shown in Table 3. For the three consecutive bits $\{d_{2i+1}, d_{2i}, d_{2i-1}\}$, the accurate and approximate encoding are listed in the last three columns in Table 3. The corresponding partial product (PP) is selected

Table 3. Approximate Partial Product in Radix-4 Booth Multipliers[22, 23]

$d_{2i+1}d_{2i}d_{2i-1}$	000	001	010	011	100	101	110	111
PP_i	0	$+1c$	$+1c$	$+2c$	$-2c$	$-1c$	$-1c$	0
[22]	0	$+1c$	$+1c$	0	0	$-1c$	$-1c$	0
[23]	0	$+1c$	$+1c$	$+1c$	$-1c$	$-1c$	$-1c$	0

from ($\pm 2c$, $\pm 1c$, 0). Here, "$\pm 2c$" means to shift the partial product. In [22, 23] parts of the encoding are simplified, as "$\pm 2c$" is turned into "$\pm c$" or "0". With this approximate method, the final multiplier shows large reduction on power, delay, area and power-delay product (PDP). Some cases show 59% reduction of PDP compared with the conventional accurate booth multiplier.

It should be noted that few researches have made efforts to propose the approximate Radix-8 booth multiplier even though more partial products reduction could be achieved compared with Radix-4. As pointed in [24], this is mainly because the encoding processing in Radix-8 requires the generation of $\pm 3x$d multiplicand (the set of $\{\pm 4x, \pm 3x, \pm 2x, \pm 1x, 0\}$ will be generated), which means that some preliminary processing should be involved and more corresponding delay or power will be consumed. In [24], this problem has been solved, as the designers turned the encoding set of $\{\pm 4x, \pm 3x, \pm 2x, \pm 1x, 0\}$ to the approximate set of $\{\pm 4x, \pm 3x, \pm 2x, \pm 1x, 0\}$, in which all the $\pm 3x$ multiplicands are approximated to the $\pm 2x$ multiplicand. Thus, the final logic-gate implementation of this approximate Radix-8 booth multiplier becomes much simpler. The experimental results show that compared with the approximate multiplier in [23], reduced energy of 22% with a comparable mean relative error distance (MRED) can be achieved.

In addition to the conventional multiplier, using a logarithmic transformation, which converts multiplication into addition, is also a novel design method as proposed in [25, 26]. It should be noted that after converting the input data to logarithmic numbers, the whole multiplication process only involves shifts and addition, which means that the area, delay and power can be reduced significantly. However, this improvement is achieved at the cost of inexact output, since the logarithmic transformation is used and approximation is unavoidable in the final results. The first logarithmic (LM) multiplier is proposed by Mitchell[25], followed by several modified designs to improve the accuracy with iterative technique or more refined approximation of logarithmic transformation. No inexact logic circuits, such as approximate adders, are used in these approximate LMs. In [26], inexact units like approximate lower-part-or adders (LOA) are used in approximate LM to further improve the accuracy and energy efficiency. The proposed LOA could compensate the error generated by the logarithmic transformation and has a much simpler computing scheme since the addition for lower bits is replaced by OR-gate without carry signal propagation. Experiments show that the proposed approximate LM in [26] has 18% lower normalized mean error distance than conventional LM with reduction of up to 37% in PDP. Similar to the design of approximate adders, new process techniques also play an essential role in the design of approximate multipliers. In [27], the authors also used new process ML gates to design the Radix-4 approximate booth multiplier. Since the new process has advantages over the conventional CMOS process in terms of power consumption, speed and area, together with the approximate design method, it is possible to design approximate multipliers with low power consumption and high performance.

2.1.3 Approximate Dividers

Compared with adders and multipliers, approximate dividers are not very widely studied. In terms of structure, dividers are more complex and require multiple clock cycles to complete the operation. However, in recent years, researchers have also applied approximate computation design methods to dividers. They have successfully traded a loss in computational accuracy for a significant gain in energy efficiency. Considering the integer division, the input data are dividend X and the non-zero divisor Y, and the output data are the quotient Q and the remainder R. The relationship of these four data can be expressed as $X = YQ + R$. Just like the adder design, a one-bit full subtractor is inevitable when the long divider is designed, which is shown in Fig.9.

$$D = X \oplus Y \oplus B_{\text{in}},$$
$$B_{\text{out}} = \overline{X \oplus Y} \times B_{\text{in}} + \overline{X}Y.$$

In [28], several approximate one-bit full subtractors are proposed as shown in Fig.10(a). In essence, some logic gates are removed and the final result may be wrong in some cases. With these accurate and approximate one-bit full subtractor cells, the non-restoring array divider cell (NADC) could be obtained as

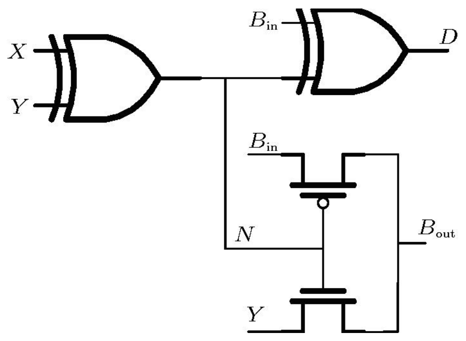

Fig.9. Accurate one-bit full subtractor.

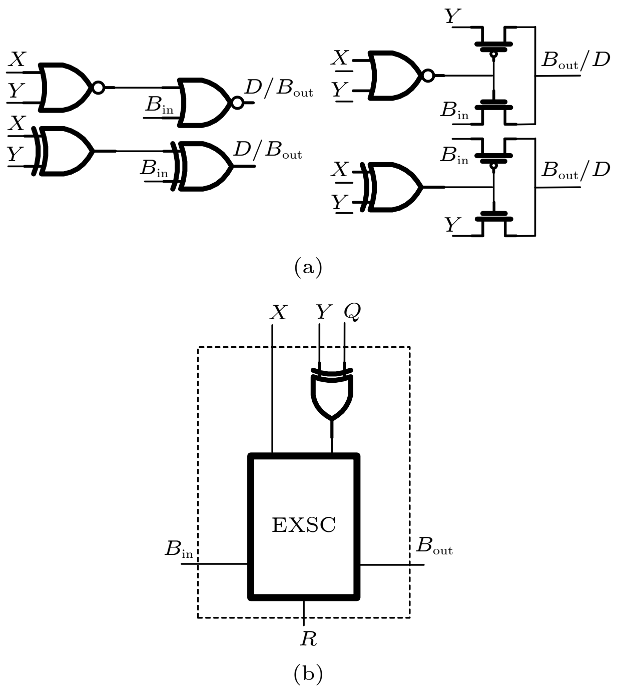

Fig.10. (a) Approximate subtractor[28]. (b) Accurate/approximate non-restoring array divider cell[28]. EXSC: an exact subtractor cell.

shown in Fig.10(b). At last, combining several NADCs, an approximate divider with long input data could be achieved, as shown in Fig.11.

As a general design principle, all exact cells can be replaced by approximate cells. However, in practical applications, the approximate cell should be applied more in the lower part of the whole array when replacing the exact cells, based on the fact that for the binary output the higher bits have more weight.

Another design point is that since the approximation unit can have different circuit structures, each circuit structure will exhibit a different error distribution. Therefore, when making a replacement, the designer needs to design a combination of approximation units with different circuit structures so that the final divider will not always have a positive or nega-

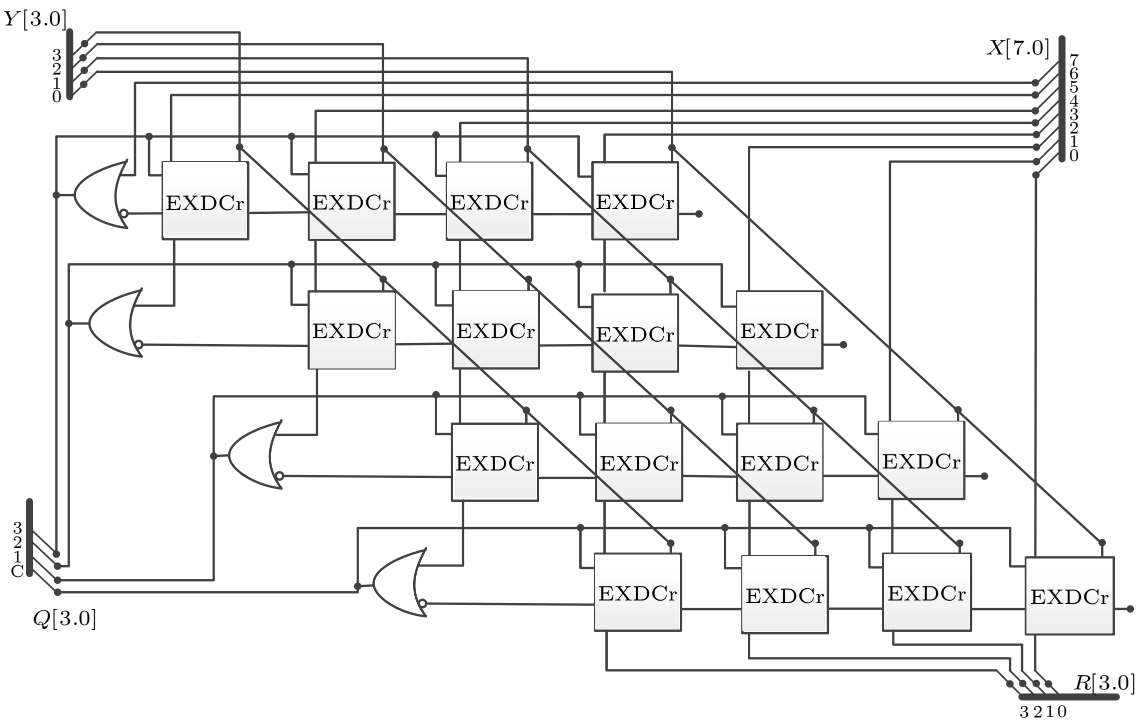

Fig.11. Approximate non-restoring divider[28]. EXDCr: an exact restoring divider cell.

tive output error but a zero-mean error distribution. For high-radix division, the technique of prescaling[29] is proposed. In [30], the authors proposed an approximate signed-digit adder cell to replace the accurate cells in [29]. Furthermore, at the array level, different configurations and replacement depths have been used so that various energy efficiency improvement and error distributions could be achieved.

2.2 Approximate Memory

For image processing and machine learning algorithms, data storage of the digital integrated circuit occupies a non-negligible proportion of the system energy consumption. For example, in video processing such as MPEG or H.264, on-chip static storage (Static Random Access Memory, SRAM) consumes a large amount of energy, about 75% of the overall motion vector estimation[31]. Meanwhile, off-chip Dynamic Random Access Memory (DRAM) accounts for 30% of the energy for entire cell phone circuit system[32]. Given the limited energy supply of most current terminal device batteries, it is important to effectively reduce the storage power consumption of on-chip SRAM and off-chip DRAM. Existing researches have conducted a series of designs using approximate storage methods in terms of application output quality and energy efficiency.

2.2.1 *Approximating SRAM*

The primary cell scheme of SRAM is shown in Fig.12[33]. This single-bit memory consists of six CMOS transistors (referred to 6T scheme). The two inverters in the middle position constitute the positive feedback, and the two NMOS transistors on both sides constitute the write and read interfaces of the memory cell. When the data is written, the WL (word line) signal is high, and then the AR and AL transistors are turned on. Thus, the input data will be writ-

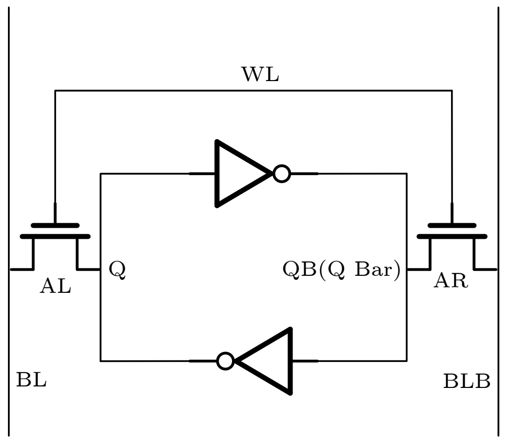

Fig.12. Circuit scheme of single-bit SRAM cell[33].

ten to the cell through the BL (bit line) and BLB (bit line bar) lines. Two situations may occur at this time. First, it is supposed that the written signal is opposite to the currently saved signal. In that case, the intermediate positive feedback inverter loop will be forcibly flipped to the desired signal value, in which both inverters will be charged-discharged and consume energy. If the written signal has the same value as the signal retained in the previous state, the write operation will not incur flip power consumption. As for reading data, the BL and BLB signal lines are first pre-charged to a high voltage, and then the pre-charge circuit will be disconnected. The load capacitors of the BL and BLB signal lines are charged and discharged through the AL-AR transistors, and then the signal value will be read. If the Q value is low, the capacitors are discharged. Conversely, if the Q value is high, the load capacitance remains unchanged. It is clear that the inverter will not flip in the reading process, in which the power consumption of the whole circuit comes only from the charging and discharging process of the load capacitor. Therefore, in the power analysis of SRAM, data writing is the primary source of power consumption of the entire on-chip storage. For on-chip memory cell circuits in Fig.12, the circuit energy is mainly determined by the following equation:

$$E = \alpha \times C \times VDD^2,$$

where α is the flip probability of the circuit, C is the equivalent capacitance, and VDD is the supply voltage. Therefore, theoretically, the designer can reduce the energy of SRAM in three ways.

However, in practical design, the equivalent capacitance is mainly related to the circuit structure, and it is challenging to improve the equivalent capacitance from the design point of view. Reducing the voltage is effective since the energy is squared with the supply voltage. However, this approach has two problems. First, after reducing the voltage, the speed of the memory cell also has to decrease. If the speed does not follow the voltage drop, fatal timing errors will occur, which will contaminate the stored data. Therefore lowering the voltage while the designer generally chooses to lower the speed hinders some high-performance circuit designs, especially for real-time processing applications. Another more problematic issue is that even in those cases where the speed requirements are not high (when processing CIF/QCIF images, the circuit can run at a lower speed of about 10 Mhz), reducing the supply voltage can satisfy both the low energy and the speed requirements. However, the memory cell faces process deviations, which may trigger logic errors with voltage drops during the read/write process. These logic errors are different from the timing errors. No matter how we decrease the speed, the logic error originating from the process deviation cannot be avoided under the low voltage conditions. The output error probability will increase with the voltage reduction.

Chang *et al.* proposed a low-voltage approximate SRAM with a hybrid scheme[33]. The authors observed that the high bits of the stored data should be accurate for applications such as image processing or machine learning. Otherwise, due to the big weights for the high bits of the stored data, an error for these bits would largely shift the whole data, which will crush the output quality. Conversely, the low bits of the stored data have a limited impact on the final output quality if errors occur with low voltage conditions.

In [33], Chang *et al.* first proposed an SRAM cell with low logic errors in a low voltage condition, as shown in Fig.12 and Fig.13. It can be seen that the core of this memory cell is still two inverter loops forming positive feedback, while two additional transistors, NB and AB, are added (referred to as 8T scheme). The readout side is effectively isolated from the whole memory cell due to NB and AB transistors, which can reduce the logic errors for data reading. At the same time, the NB transistor can effectively reduce the driving capability required for the positive feedback circuit to flip during the write operation. Therefore, this circuit scheme can reduce the probability of wrong flips for the SRAM at low voltage.

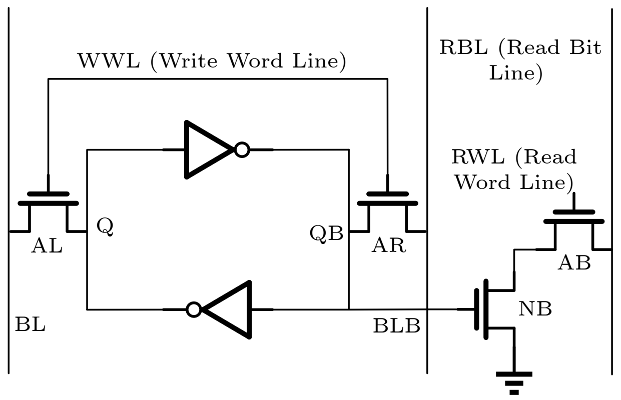

Fig.13. Circuit scheme of 8T SRAM cell.

At last, using both 8T and 6T memory cells, the raw input data could be divided into two parts. The data will be stored using the 8T scheme for the high bits, and the 6T scheme will be used for the low bits. Two main design metrics determine the ratio of 8T and 6T. First, the output quality needs to meet the lower limit set by the designer at the beginning. Second, the energy consumption should be reduced as much as possible while the output quality is satisfied.

As shown in Table 4, a standard test video "AKIY" in CIF format with 50 frames is used to verify the proposed hybrid SRAM in [33]. The video data will be processed with the hybrid SRAM, and then the data will be fed into the MPEG decoder. Finally, the peak-signal-to-noise ratio (PSNR) between approximate SRAM output and exact SRAM output will be achieved. Since the pixel data has eight bits, as shown in Table 4, "0-bit 8" means that all the data bits are stored by 6T memory cells with three different supply voltages of 600 mV, 700 mV and 800 mV, respectively. The other settings are similar, i.e., "6-bit 8" means that the first six bits of 8-bit raw data are stored by 8T memory cells, and the last two bits are stored by 6T memory cells. From the experimental results, we can see that at 800 mV, if the 6T memory cells are used for all data bits, the final output quality is not much degraded due to the small logic error with this voltage. However, as the voltage keeps dropping, the output quality of all 6T memory cells for storage decreases from 23.48 dB to 14.75 dB, which is unacceptable in practical applications. At the same time, it can be seen that the output quality keeps improving with the increasing proportion of 8T memory cells under the 600 mV supply voltage. If the designer allows less than 1 dB PSNR loss, the "4-bit 8" storage setting can meet this requirement, and at this time, the power consumption could be reduced significantly due to the lower voltage.

Table 4. PSNR (dB) with Different Ratios of 8T and 6T at Various Voltages[33]

VDD (V)	0-Bit 8T	1-Bit 8T	2-Bit 8T	3-Bit 8T	4-Bit 8T	5-Bit 8T	6-Bit 8T	7-Bit 8T	8-Bit 8T
0.6	14.75	19.26	21.86	22.96	23.96	23.42	23.49	23.55	23.61
0.7	21.41	22.92	23.40	23.54	23.57	23.59	23.60	23.60	23.61
0.8	23.48	23.58	23.60	23.61	23.61	23.61	23.61	23.61	23.61

2.2.2 *Approximating DRAM*

Off-chip DRAM also has a large amount of energy consumption in the circuit system. A single-bit dynamic memory cell[34] is shown in Fig.14, which consists of capacitors, MOS transistors, and a sensitive amplifier. When the row and column addresses are set to "1", the two MOS transistors are turned on, and

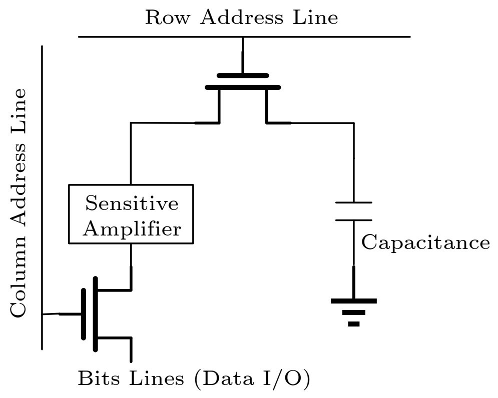

Fig.14. Circuit scheme of DRAM cell[34].

the input data will charge or discharge the capacitor through the sensitive amplifier. When the written data is "0", the capacitor will be discharged. The capacitor will be charged when the written data is "1". However, as the capacitor itself has a leakage problem, it needs to be charged again by the refresh circuit after some time. Therefore, as pointed out in [35], the power consumption of DRAM is proportional to the refresh frequency of the entire memory device and the number of high voltage bits stored.

Liu *et al.* proposed a multi-stage refresh frequency for approximate DRAM[36]. The authors also followed the principle that the high bits of the stored data cannot be contaminated, and the refresh frequency of high bits is kept constant to ensure correctness. For the lower bits, the refresh frequency is reduced at different levels. It should be noted that reducing the refresh frequency only causes errors for the bits stored as high voltage (logic "1") because these high voltage bits may not be refreshed in time with reduced frequency. Based on this idea, the refresh frequency of the DRAM can be scheduled according to different applications and output quality requirements, thus reducing the power consumption while losing some of the output quality. The main problem in this approach is the need to modify the original DRAM refresh control system, and the additional overhead caused by this modification cannot be neglected. In [37], researchers also adopted the same approach to reduce the power consumption of DRAM by using dynamic data truncation to reduce the amount of data storage. The storage controller truncates the raw data for specific applications and output quality requirements. Output quality checks are performed at regular intervals during the runtime phase to generate feedback and adjust the number of bits truncated by the memory controller. However, there is a considerable loss of information due to simple data truncation, leading to significant degradation of the final output quality. Therefore, this design method has some limitations in practical application.

2.2.3 Approximating Memories Based on New Process

Unlike SRAM and DRAM memory under the traditional CMOS process, memory devices composed of various new materials have also received much attention. Non-volatile random-access memory[38] can keep the data when power is lost, which is a significant advantage for conventional memory, especially in the case of an unstable power supply system. The problem with non-volatile memory is that the device may fail within a short period if the memory is read and written frequently. Researchers can use methods such as data compression to reduce the number of data reads and writes and extend the life of the device. In recent years, memristors have also received great attention. Resistive Random Access Memory (RRAM) allows for high-performance, large-scale computation. Designers have used its cross-switching matrix for convolutional neural network approximation[39], which has significantly improved energy efficiency. However, due to a large amount of ADC interfaces, additional overhead to the design is inevitable. Moreover, the RRAM process is very mature. Large process deviations and a wide range of threshold voltage shifts make it limited in practice. Similar to RRAM, Phase Change Memory (PCM)[40] has the distinct feature of having several different intermediate states and a large storage capacity. However, PCM's fatal problem is to update the data because it requires multiple writes to this memory device. Otherwise, the data written will generate some logical errors. However, multiple writes usually incur a sizeable additional overhead; therefore the authors[40] also apply the principle of approximate computation to ensure the correctness of high bits data using multiple writes while reducing the number of writes for the low bits data. Similar to RRAM, PCM's biggest challenge is the threshold shift in the long-term operation, which needs to be addressed in large-scale applications.

2.3 Software Level

2.3.1 Loop Perforation

Loop perforation is an algorithm-based approximation technique that trades computational precision for performance efficiency, by skipping specific iterations in a loop, contributing to the reduced computational cost and a significant performance gain without executing all code iterations. For example, when performing finite element analysis, the more the partitioned meshes are, the longer the execution time is, and the higher the energy consumption is. Hence, the researcher has ability to choose to skip some of the mesh blocks with repetitive or unimportant roles to acquire the execution time advantage and performance improvement. This approximation technique[41] is applied to the computational model of a generic algorithm by sequentially perforating the loop with a given perforation scheme (that is, the fraction of iterations to be skipped, referred to as the perforation rate in the literature), using representative inputs for the perforation computation, and evaluating the computational system outputs (which are probably not available). Finally, extremely erroneous iterations are filtered out, and the system is crashed, resulting in the critical loops (called unapproximable or non-adjustable loops) featuring a significant negative impact such as fatal errors, system crashes, an apparent decrease in inefficiency, or an evident increase in execution time due to the filtered-out perforation. The remaining terms that can be perforated (namely, adjustable loops) are approximated. Moreover, two algorithms have been investigated in the literature to explore a perfect balance between efficiency and accuracy. The first algorithm adopts a specific strategy, namely using multiple perforation rates to filter out the adjustable cycles. Thus, the efficiency and the accuracy of the Pareto-optimal variants are derived between the perforation rate and the adjustable cycles. The other algorithm adopts a greedy strategy, namely using heuristic scoring metrics to prioritize the cycles/perforation rates, so as to seek the maximum operating efficiency within a given accuracy error. For a range of applications, loop perforation can typically improve performance up to more than twice as much while incurring no more than a 10% loss in accuracy.

Mainstream parallel computing frameworks such as OpenMP and OpenACC are employed to accelerate the circularly parallelized applications. OpenMP is suitable for parallel programming on multicore CPU machines, and OpenACC supports CPU/GPU work. In the accelerated applications using such computing frameworks, many loop iterations are started as threads on the accelerator, which fits well with loop perforation applications. A new instruction[42] was utilized in OpenACC to trade off performance and accuracy by perforating loop iterations.

Loop perforation is an algorithmic software level based technique, which is only applicable to cyclic iterative code structures. In other words, loop-through techniques can be applied in software and programmable hardware codes. As for the FPGA side, a cyclic iterative code structure can be multiple circuits executing in parallel or a constantly re-executing circuit. The impact of implementing loop-through on the software side is different in comparison with the FPGA side. The software side primarily influences the arithmetic power consumption and execution time of the application, while the FPGA side impacts the area and energy cost.

2.3.2 Data Precision Reduction

The most straightforward concept of data precision (the number of data bits) reduces the memory footprint and hence exchanges the cost of precision reduction for memory consumption and performance improvement. Data precision reduction can be implemented at the software level in various ways to constrain precision and achieve performance goals. Reducing the bit width used for data representation is one of the most prevalent approaches. For example, DoD *et al.*[43] introduced a dynamic program analysis tool called Precimonious to assist developers to tune the precision of floating-point programs. Precimonious first creates a search space under the project containing all the local variables of the functions accessible from the main static. After such an operation, each variable in the search file is associated with a set of types. Subsequently, each of the type sets is refined by using an algorithmic iteration that considers a pair of the highest and second highest precision available and then identifies the set of floating-point variables assigned with the highest precision, which would be ignored in the next iteration. Furthermore, valid type configurations are identified after the generation of a program variant that can automatically reflect the type configuration. The implementation has a performance advantage over most programs when instantiated with lower precision types. An energy-aware hybrid precision selection framework called EHPS[44] was proposed to reduce the consumption of shaders in the mobile GPUs. In comparison with traditional reduction mechanisms, EHPS combines a traditional mechanism consisting of fixed-point and reduced floating-point precision with a contour-based precision selection mechanism that maximizes energy savings. Through this mechanism, the range of values is shortened from an entire precision floating point, and the reduction in image data precision contributes to an overall reduction in image quality. More specifically, it makes sense to trade accuracy for performance as long as the degradation of image quality is kept within the range that is not perceptible to the human eye or acceptable to the user.

Implementing specific algorithms efficiently in some classification problem scenarios is of overriding importance. Support vector machines (SVM) represent a robust nonlinear classifier that is possibly not efficiently implemented in the SVM classification stage ascribed to arithmetic and energy limitations in some complex scenarios. One possible approach is to reduce the working accuracy of SVM to adapt to working scenarios where arithmetic power and energy consumption do not support it. The relationship between SVM classification accuracy and floating-point arithmetic accuracy has been investigated[45], specifically, researchers adapted based on the perturbation bounds, and experiments were performed with three publicly available benchmark datasets. The results demonstrated much room for a substantial reduction in the working accuracy before the SVM classification accuracy reaches the loss limit level. Moreover, data accuracy reduction also has wide applications for neural networks (NNs). For example, deep networks can be trained with only fixed-point numbers with less bit-width and no degradation in classification accuracy by approximating the bit-width of the data with a random probability. This application aspect will be detailed in the survey.

2.4 Application Level

In this subsection, two mainstream approximation methods in the field of machine learning will be introduced in detail. In machine learning, neural networks are increasingly used for tasks such as recognition, classification, and segmentation. The neural network technology, firstly called perceptron, consisted of a simple three-layer structure of an input layer, an implicit layer, and an output layer. Later, multilayer perceptron was proposed to solve the defect of not being able to fit the heterogeneous logic and also brought inspiration to the development of neural networks, where the number of layers directly determined the ability to portray reality. Pre-training is used to alleviate the problem of local optimality. Instead of using the sigmoid function, transfer functions such as ReLU or max were used to overcome part of the gradient disappearance problem, gradually leading to today's DNNs with more implicit layers, especially the recent emergence of deep residual learning to further avoid gradient disappearance. The increase in network layers makes NNs more capable of representation and highly complex. Besides, spatially deep convolutional neural networks (CNNs) update the structure of neural networks by adding convolutional kernels. However, the disadvantages of highly complex large-scale neural networks should not be ignored as computation or power consumption can be exceptionally high. Introducing approximate computing into neural networks is a feasible approach, which raises the challenging question of how to employ approximate computing in NNs systematically. Related researches have proposed many representative solutions to employ approximate computing from the modeling phase to the inference phase, achieving low power and high energy efficiency.

In the training phase of neural networks, the backpropagation algorithm is currently the most common and efficient training algorithm. After the forward transmission process of NNs training, the error between the output result of the output layer and the actual output result is calculated, and the error is redistributed forward into the network until it is propagated to the input layer. The network parameters are generally initialized randomly, adjusted according to the backpropagation error, and iterated continuously until convergence. Such properties of backpropagation algorithms provide a measure of the error of each neuron on the network output. For example, Venkataramani *et al.* proposed a backpropagation-based approximate neural network (AxNN) design method that utilizes the backpropagation property to characterize the importance of each neuron in the NNs and determine the neurons more sensitive to the quality of the output and the insensitive neurons (called resilient neurons in their description)[46]. The identified neurons with less impact on the quality of the work were replaced with the resilient neurons using approximate neurons, where precision scaling techniques implemented the approximate neurons. Finally, incremental retraining of the AXNN was performed. Backpropagation, in essence, can autonomously repair the errors in the network and can benefit from the errors generated when using the approximation technique.

2.4.1 *Parameter Quantization*

In DNNs model training, model parameters such as weights of the NNs and bias data types are typically stored in computations with double-precision (FP64), single-precision (FP32), and half-precision (FP16) data types. Today, most DNNs applications use FP32 for handling training and inference workloads. Quantization approximates a continuous signal through a set of discrete symbols. The basic idea of model quantization is to replace the original floating-point precision with lower precision. Low-bit parameter quantization could result in significant reductions in bandwidth, energy consumption, and chip area during training. The smaller the bits of the model are, the smaller the model storage is, and the more significant the execution speedup to occur is. The central challenge is to weaken the representation accuracy without significant degradation of the model accuracy, suggesting that the usability of the gradient descent algorithm is maintained during backpropagation. The classification accuracy did not significantly degrade when the floating-point parameters were approximated to a 16-bit vast fixed-point number (int16) representation using a random rounding method, quantized and trained on the DNNs[47]. The bit width of the parameters was further reduced. With the block-based accumulation and floating-point random rounding, training was performed using 8-bit wide floating-point precision (FP8). Additionally, 8-bit comprehensive fixed-point numeric representation was employed in training, while retraining was required, and accuracy was not guaranteed in most cases after int8 quantization[48]. Zhu *et al.* proposed a unified int8 training framework[49] to stabilize int8 training using generic techniques such as reducing gradient direction bias and avoid illegal gradient updates along the wrong direction. A CNN approximation framework, called Ristretto, constructed by Gysel *et al.*[50], allows empirical studies on the trade-off among various digit representations and word width choices, as well as the classification accuracy of the model, so as to fine-tune the model parameters after int8 quantization and recover a portion of the model accuracy loss due to approximation. Designs were also presented for the extreme quantization of the last 4 bits and less than 4 bits ($\leqslant$ INT4). A practical 4-bit post-training quantization method[51] was provided to quantify activations and weights. Moreover, three novel methods of analytical clipping of integer quantization (ACIQ), bit allocation strategy, and bias correction were introduced to minimize tensor-level quantization errors. The INT4 quantization fine-tuning phase has many instabilities. Various methods to overcome this difficulty were proposed, such as using a small subset of the training set to calibrate the activation size and discarding outlier activations based on percentile and clamped quantization activations and gradients. A compromise was achieved for the problem of over-quantized parameters and unstable models, where the training employed two or more different bit-width types in the model simultaneously, namely, mixed-precision training. For example, DNNs training using a hybrid FP8 format was demonstrated[52], which uses two FP8 formats for forwarding and backward propagation. A mixed-precision scheme was adopted[53], where FP16 is taken for forwarding propagation and gradient computation, while FP32 stores the gradients of the network parameters (which is called weight backup) and effectively mitigates the gradient disappearance in low-bit quantization training.

2.4.2 Model Pruning

Model pruning is a commonly-used model approximation method to efficiently generate models with smaller sizes, higher memory utilization, lower energy consumption, and faster inference with less accuracy loss. Model pruning in NNs is inspired by synaptic pruning in the human brain, which is the complete decline and death of axons and dendrites. Similar work can be conducted in NNs, and the basic idea is to prune the least important parts. For example, neurons in a network can be ranked according to their contribution, and then a smaller and faster network can be obtained by removing the lower-ranked neurons. The design concept of model pruning has been researched for a long time. For example, a magnitude-based pruning method was proposed to minimize the number of hidden units by applying a weight decay to each remote unit in the network concerning its absolute value[54]. OBD and OBS methods were proposed[55] to measure the importance of weights in the network based on the second-order derivatives of the loss function considering the weights. Model pruning is only gradually emphasized after DNNs are widely used and the number of network layers is deepened. These design methods have been exerting profound influence on the development of model pruning later. Model pruning is unavoidable in the era of widespread DNNs use. For example, Google explored the performance comparison between large sparse models and small dense models[56] where large sparse models consistently outperformed small dense models in a wide range of neural network architectures.

The fine-grained pruning approach is straightforward. As shown in Fig.15, first, a baseline model is trained, and then the magnitude of the weights is sorted to remove the connection below a predefined threshold, resulting in a pruned network. Finally, the pruned network is fine-tuned, and the execution continues from the previous step until the termination condition is satisfied. One of the critical issues is how to evaluate the importance of these connections. Thus, NVIDIA proposed a magnitude-based approach to evaluate the connection importance[57].

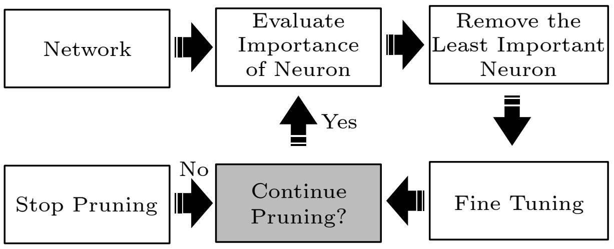
Network
Evaluate Importance of Neuron
Remove the Least Important Neuron
Yes
No
Stop Pruning
Continue Pruning?
Fine Tuning

Fig.15. Pruning process.

Compared with fine-grained pruning, coarse-grained pruning will be more effective in obtaining small, sparse models that do not require specialized algorithmic support. Coarse-grained pruning can be performed on filters or feature channels. One approach to featuring channel pruning is to evaluate the effectiveness of a channel in conjunction with constraining some channels to make the model sparse. In [58], extensive and large networks are used as input models, and during training, channels with higher sparsity are automatically identified and removed, resulting in compact models with considerable accuracy. The scaling factor in batch normalization is applied to crop the unimportant channels. In [59], an end-to-end random training approach is used to force the output of specific channels to be constant and then remove these constant channels in the original neural network by adjusting the bias of its influence layer in order to fine-tune the generated compact model quickly.

3 Limitations of Approximate Computing

As mentioned earlier, approximate computing, a paradigm shifted from traditional accurate processing, has potential usage at all stack levels of computing systems, especially for large-scale centralized and distributed edge computing. However, approximate computing also has limitations. In the specific implementation of approximate computing, fault tolerance and security vulnerability are two necessary conditions to confirm whether approximations can be applied in the practical applications.

3.1 Fault Tolerance of a System

Fault tolerance of a system is generally defined as the ability of the system to execute a given algorithm correctly without affecting the accuracy of the system or within an acceptable level of output quality when approximations or errors are introduced in the inputs or intermediate operations of the computing system. Approximate computing is based on the fault tolerance of the system. Thus, an essential condition for applications with approximate computing is whether the computing system is fault-tolerant. The computing system is considered unsuitable for the approximation mechanism if it is not fault-tolerant. In general, safety-critical systems are less fault-tolerant, and therefore the use of approximate computing to such systems is sophisticated. Moreover, though the fault tolerance of the computing system satisfies the approximation mechanism, there is another situation where the approximate computing is also not suitable, and where the introduction of the approximate computing brings adverse effects such as an obvious increase in execution time, a significant increase in energy consumption, and an unacceptable decrease in performance, which are all contrary to the original intention of using the approximate computing.

3.2 Unpredictable Security Vulnerability

Approximate computing techniques carry an intrinsic uncertainty, and to some extent probably introduce new unpredictable security vulnerability to the computing system. The following reasons should be considered. First, in a computing system with approximate modules, there is a clear boundary among approximate modules and other exact modules in most cases. Supposing that there is no unique hiding mechanism for approximate modules. In this case, this boundary works as a practical guide for attackers to attack the computing system, allowing attackers to recognize vulnerable parts of the computing system faster and provide a new attack surface to damage the computing system. In terms of approximate adders, for example, the approximate operation can dramatically change the transition probability of specific signals, resulting in easier-to-detect security vulnerability by attackers. Second, approximate modules are generally fault-tolerant. Therefore, distinguishing whether the error is caused by the approximate computing or carefully designed by attackers is arduous, and therefore attackers can easily falsify the attack into an approximation in the module.

4 Promises and Challenges of Approximate Computing

Approximate computing is a widely applicable and excellent paradigm in different applications. Although approximate computing is not mature enough, there is a glimpse of the unlimited potential of approximate computing concerning energy efficiency and high performance. Approximation computing will be confronted with opportunities and many challenges in the coming years.

4.1 Opportunities to Come

Apparently approximate computing is applicable at all stack levels of the system and have evaluated representative techniques at each stack level. When most of the techniques at each stack level become matured, a natural trend to investigate compounding effects by applying multiple techniques in the system will emerge. For the purpose of exploiting such compounding effects, considering holistically at multiple levels of the system stack and designing new reasonable and effective accelerator cannot be ignored. For example, IBM proposed the TeraOPS deep learning processor core in 2018[60]. This is an AI accelerator chip that compounds approximation techniques at multiple stack levels to achieve performance and area advantages compared with other high-performance AI chips. In [61], the researchers proposed a resource-oriented high-level synthesis (HLS) method, in which heterogeneous resource constraints can be defined with minimizing the output error. The proposed method is applied to approximate designs in FPGAs. With this HLS method, the designers could exploit the resources in FPGA more efficiently with certain approximation.

4.2 Challenges to Be Faced

Approximation techniques introduce security vulnerabilities, which are a significant concern for the continued development of approximation techniques in the future. The security threats such as uncertain results of approximation execution during approximation execution, may be indistinguishable from data tampered with by malicious attacks presented in Section 3. More effective solutions should be proposed in the research related to the security applications of approximate computing. In [62], the researchers proposed security protection methods using redundant number representation, most significant digit first arithmetic, and algorithmic approximation analysis. The exemplary stationary iterative solver is used to hide information. Furthermore, applying approximation techniques for security-critical and high-cost systems is a promising direction for extensive future research.

5 Conclusions

This paper discusses the concept of approximate computing and its various applications. It provides a broad overview of the field, starting from the design of approximate arithmetic units and ending with high-level applications of approximate computing. One of the main conclusions of the survey is that approximate computing is a promising approach to improving the performance, energy efficiency, and cost of computing systems. The paper shows that there are many opportunities for approximate computing in different areas, including machine learning, signal processing, and multimedia processing. Another conclusion is that the design of approximate arithmetic units is a critical aspect of approximate computing. The paper presents several techniques for designing approximate arithmetic units. We also discussed the advantages and limitations of each technique and highlighted the importance of choosing the right approach for a particular application. Furthermore, the paper emphasizes the need for tools and methodologies to support the design and implementation of approximate computing systems.

References

[1] Xu Q, Mytkowicz T, Kim N S. Approximate computing: A survey. *IEEE Design & Test*, 2016, 33(1): 8–22. DOI: 10.1109/MDAT.2015.2505723.

[2] Zervakis G, Saadat H, Amrouch H, Gerstlauer A, Parameswaran S, Henkel J. Approximate computing for ML: State-of-the-art, challenges and visions. In *Proc. the 26th Asia and South Pacific Design Automation Conference* (*ASP-DAC*), Jan. 2021, pp.189–196. DOI: 10.1145/3394885.3431632.

[3] Jiang H L, Santiago F J H, Mo H, Liu L B, Han J. Approximate arithmetic circuits: A survey, characterization, and recent applications. *Proceedings of the IEEE*, 2020, 108(12): 2108–2135. DOI: 10.1109/JPROC.2020.3006451.

[4] Amanollahi S, Kamal M, Afzali-Kusha A, Pedram M. Circuit-level techniques for logic and memory blocks in approximate computing systems. *Proceedings of the IEEE*, 2020, 108(12): 2150–2177. DOI: 10.1109/JPROC.2020.3020792.

[5] Cheemalavagu S, Korkmaz P, Palem K V *et al.* A probabilistic CMOS switch and its realization by exploiting noise. In *Proc. IFIP International Conference on VLSI*, Oct. 2005, pp.535–541.

[6] Gupta V, Mohapatra D, Raghunathan A, Roy K. Low-power digital signal processing using approximate adders. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 2013, 32(1): 124–137. DOI: 10.1109/TCAD.2012.2217962.

[7] Kim Y, Zhang Y, Li P. An energy efficient approximate adder with carry skip for error resilient neuromorphic VLSI systems. In *Proc. the 2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov. 2013, pp.130–137. DOI: 10.1109/ICCAD.2013.6691108.

[8] Zhu N, Goh W L, Wang G, Yeo K S. Enhanced low-power high-speed adder for error-tolerant application. In *Proc. the 2010 International SoC Design Conference*, Nov. 2010, pp.323–327. DOI: 10.1109/SOCDC.2010.5682905.

[9] Lin I C, Yang Y M, Lin C C. High-performance low-power carry speculative addition with variable latency. *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 2015, 23(9): 1591–1603. DOI: 10.1109/TVLSI.2014.2355217.

[10] Hu J J, Li Z J, Yang M, Huang Z X, Qian W K. A high-accuracy approximate adder with correct sign calculation. *Integration*, 2019, 65: 370–388. DOI: 10.1016/j.vlsi.2017.09.003.

[11] Yang X H, Xing Y, Qiao F, Yang H Z. Multistage latency adders architecture employing approximate computing. *Journal of Circuits, Systems and Computers*, 2017, 26(3): 1750039. DOI: 10.1142/S0218126617500396.

[12] Zhang T T, Liu W Q, McLarnon E, O'Neill M, Lombardi F. Design of majority logic (ML) based approximate full adders. In *Proc. the 2018 IEEE International Symposium on Circuits and Systems* (*ISCAS*), May 2018. DOI: 10.1109/ISCAS.2018.8350962.

[13] Liang J H, Han J, Lombardi F. New metrics for the reliability of approximate and probabilistic adders. *IEEE Trans. Computers*, 2013, 62(9): 1760–1771. DOI: 10.1109/TC.2012.146.

[14] Niharika A, Ramesh M K. 16×16 modified booth multiplier implementation using Wallace tree structures. *Journal of Signal Processing*, 2022, 8(1): 16–21.

[15] Kulkarni P, Gupta P, Ercegovac M. Trading accuracy for power with an underdesigned multiplier architecture. In *Proc. the 24th Internatioal Conference on VLSI Design*, Jan. 2011, pp.346–351. DOI: 10.1109/VLSID.2011.51.

[16] Rehman S, El-Harouni W, Shafique M, Kumar A, Henkel J, Henkel J. Architectural-space exploration of approximate multipliers. In *Proc. the 2016 IEEE/ACM International Conference on Computer-Aided Design* (*ICCAD*), Nov. 2016. DOI: 10.1145/2966986.2967005.

[17] Waris H, Wang C H, Xu C Y, Liu W Q. AxRMs: Approximate recursive multipliers using high-performance building blocks. *IEEE Trans. Emerging Topics in Computing*, 2022, 10(2): 1229–1235. DOI: 10.1109/TETC.2021.3096515.

[18] Mahdiani H R, Ahmadi A, Fakhraie S M, Lucas C. Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications. *IEEE Trans. Circuits and Systems I: Regular Papers*, 2009, 57(4): 850–862. DOI: 10.1109/TCSI.2009.2027626.

[19] Baran D, Aktan M, Oklobdzija V G. Energy efficient implementation of parallel CMOS multipliers with improved compressors. In *Proc. the 16th ACM/IEEE International Symposium on Low-Power Electronics and Design*, Aug. 2010, pp.147–152. DOI: 10.1145/1840845.1840876.

[20] Zendegani R, Kamal M, Bahadori M, Afzali-Kusha A, Pedram M. RoBA multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing. *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 2017, 25(2): 393–401. DOI: 10.1109/TVLSI.2016.2587696.

[21] Narayanamoorthy S, Moghaddam H A, Liu Z H, Park T, Kim N S. Energy-efficient approximate multiplication for digital signal processing and classification applications. *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 2015, 23(6): 1180–1184. DOI: 10.1109/TVLSI.2014.2333366.

[22] Liu W Q, Qian L Y, Wang C H, Jiang H L, Han J, Lombardi F. Design of approximate radix-4 booth multipliers for error-tolerant computing. *IEEE Trans. Computers*, 2017, 66(8): 1435–1441. DOI: 10.1109/TC.2017.2672976.

[23] Venkatachalam S, Adams E, Lee H J, Ko S B. Design and analysis of area and power efficient approximate booth multipliers. *IEEE Trans. Computers*, 2019, 68(11): 1697–1703. DOI: 10.1109/TC.2019.2926275.

[24] Waris H, Wang C H, Liu W Q. Hybrid low radix encoding-based approximate booth multipliers. *IEEE Trans. Circuits and Systems II: Express Briefs*, 2020, 67(12): 3367–3371. DOI: 10.1109/TCSII.2020.2975094.

[25] Mitchell J N. Computer multiplication and division using binary logarithms. *IRE Trans. Electronic Computers*, 1962, EC-11(4): 512–517. DOI: 10.1109/TEC.1962.5219391.

[26] Liu W Q, Xu J H, Wang D Y, Wang C H, Montuschi P, Lombardi F. Design and evaluation of approximate logarithmic multipliers for low power error-tolerant applications. *IEEE Trans. Circuits and Systems I: Regular Papers*, 2018, 65(9): 2856–2868. DOI: 10.1109/TCSI.2018.2792902.

[27] Zhang T T, Jiang H L, Mo H, Liu W Q, Lombardi F, Liu L B, Han J. Design of majority logic-based approximate booth multipliers for error-tolerant applications. *IEEE Trans. Nanotechnology*, 2022, 21: 81–89. DOI: 10.1109/TNANO.2022.3145362.

[28] Chen L B, Han J, Liu W Q, Lombardi F. On the design of approximate restoring dividers for error-tolerant applications. *IEEE Trans. Computers*, 2016, 65(8): 2522–2533. DOI: 10.1109/TC.2015.2494005.

[29] Ercegovac M D, Lang T, Montuschi P. Very-high radix division with prescaling and selection by rounding. *IEEE Trans. Computers*, 1994, 43(8): 909–918. DOI: 10.1109/12.295853.

[30] Chen L B, Lombardi F, Montuschi P, Han J, Liu W Q. Design of approximate high-radix dividers by inexact binary signed-digit addition. In *Proc. the on Great Lakes Symposium on VLSI 2017*, May 2017, pp.293–298. DOI: 10.1145/3060403.3060404.

[31] Lin C P, Tseng P C, Chiu Y T, Lin S S, Cheng C C, Fang H C, Chao W M, Chen L G. A 5mW MPEG4 SP encoder with 2D bandwidth-sharing motion estimation for mobile applications. In *Proc. the 2006 IEEE International Solid State Circuits Conference-Digest of Technical Papers*, Feb. 2006, pp.1626–1635. DOI: 10.1109/ISSCC.2006.1696217.

[32] Carroll A, Heiser G. An analysis of power consumption in a smartphone. In *Proc. the 2010 USENIX Conference on USENIX Annual Technical Conference*, Jun. 2010.

[33] Chang I J, Mohapatra D, Roy K. A priority-based 6T/8T hybrid SRAM architecture for aggressive voltage scaling in video applications. *IEEE Trans. Circuits and Systems for Video Technology*, 2011, 21(2): 101–112. DOI: 10.1109/TCSVT.2011.2105550.

[34] Zhou N, Qiao F, Yang H Z, Wang H. Low-power off-chip memory design for video decoder using embedded bus-invert coding. In *Proc. the 10th International Symposium on Autonomous Decentralized Systems*, Mar. 2011, pp.251–255. DOI: 10.1109/ISADS.2011.33.

[35] Joo Y, Choi Y, Shim H. Energy exploration and reduction of SDRAM memory systems. In *Proc. the 2002 Design Automation Conference*, Jun. 2002, pp.892–897. DOI: 10.1109/DAC.2002.1012748.

[36] Liu S, Pattabiraman K, Moscibroda T, Zorn B G. Flikker: Saving DRAM refresh-power through critical data partitioning. In *Proc. the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2011, pp.213–224. DOI: 10.1145/1950365.1950391.

[37] Tian Y, Zhang Q, Wang T, Yuan F, Xu Q. ApproxMA: Approximate memory access for dynamic precision scaling. In *Proc. the 25th Edition on Great Lakes Symposium on VLSI*, May 2015, pp.337–342. DOI: 10.1145/2742060.2743759.

[38] Shiga H, Takashima D, Shiratake S I, Hoya K, Miyakawa T, Ogiwara R, Fukuda R, Takizawa R, Hatsuda K, Matsuoka F, Nagadomi Y, Hashimoto D, Nishimura H, Hio-

ka T, Doumae S, Shimizu S, Kawano M, Taguchi T, Watanabe Y, Fujii S, Ozaki T, Kanaya H, Kumura Y, Shimojo Y, Yamada Y, Minami Y, Shuto S, Yamakawa K, Yamazaki S, Kunishima I, Hamamoto T, Nitayama A, Furuyama T. A 1.6 GB/s DDR2 128 Mb chain FeRAM with scalable octal bitline and sensing schemes. *IEEE Journal of Solid-State Circuits*, 2010, 45(1): 142–152. DOI: 10.1109/JSSC.2009.2034414.

[39] Li B X, Xia L X, Gu P, Wang Y, Yang H Z. Merging the interface: Power, area and accuracy co-optimization for RRAM crossbar-based mixed-signal computing system. In *Proc. the 52nd ACM/EDAC/IEEE Design Automation Conference*, Jun. 2015. DOI: 10.1145/2744769.2744870.

[40] Nelson J, Sampson A, Ceze L. Dense approximate storage in phase-change memory. In *Proc. the Wild and Crazy Ideas w/International Conference on Architectural Support for Programming Languages and Operating Systems* (*WACI w/ASPLOS*), Mar. 2011.

[41] Sidiroglou-Douskos S, Misailovic S, Hoffmann H, Rinard M. Managing performance vs. accuracy trade-offs with loop perforation. In *Proc. the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, Sept. 2011, pp.124–134. DOI: 10.1145/2025113.2025133.

[42] Lashgar A, Atoofian E, Baniasadi A. Loop perforation in OpenACC. In *Proc. the 2018 IEEE International Conference on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/-BDCloud/SocialCom/SustainCom)*, Dec. 2018, pp.163–170. DOI: 10.1109/BDCloud.2018.00036.

[43] Rubio-González C, Nguyen C, Nguyen H D, Demmel J, Kahan W, Sen K, Bailey D H, Iancu C, Hough D. Precimonious: Tuning assistant for floating-point precision. In *Proc. the International Conference on High Performance Computing, Networking, Storage and Analysis*, Nov. 2013. DOI: 10.1145/2503210.2503296.

[44] Hsiao C C, Chu S L, Chen C Y. Energy-aware hybrid precision selection framework for mobile GPUs. *Computers & Graphics*, 2013, 37(5): 431–444. DOI: 10.1016/j.cag.2013.03.003.

[45] Lesser B, Mücke M, Gansterer W N. Effects of reduced precision on floating-point SVM classification accuracy. *Procedia Computer Science*, 2011, 4: 508–517. DOI: 10.1016/j.procs.2011.04.053.

[46] Venkataramani S, Ranjan A, Roy K, Raghunathan A. AxNN: Energy-efficient neuromorphic systems using approximate computing. In *Proc. the 2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, Aug. 2014, pp.27–32. DOI: 10.1145/2627369.2627613.

[47] Gupta S, Agrawal A, Gopalakrishnan K, Narayanan P. Deep learning with limited numerical precision. In *Proc. the 32nd International Conference on Machine Learning*, Jul. 2015, pp.1737–1746.

[48] Krishnamoorthi R. Quantizing deep convolutional networks for efficient inference: A whitepaper. arXiv: 1806.08342, 2018. https://arxiv.org/abs/1806.08342, April 2023.

[49] Zhu F, Gong R H, Yu F W, Liu X L, Wang Y F, Li Z L, Yang X Q, Yan J J. Towards unified INT8 training for convolutional neural network. In *Proc. the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2020, pp.1966–1976. DOI: 10.1109/CVPR42600.2020.00204.

[50] Gysel P, Pimentel J, Motamedi M, Ghiasi S. Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks. *IEEE Trans. Neural Networks and Learning Systems*, 2018, 29(11): 5784–5789. DOI: 10.1109/TNNLS.2018.2808319.

[51] Banner R, Nahshan Y, Soudry D. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Proc. the 33rd International Conference on Neural Information Processing Systems*, Dec. 2019, pp.7950–7958.

[52] Sun X, Choi J, Chen C Y, Wang N G, Venkataramani S, Srinivasan V V, Cui X D, Zhang W, Gopalakrishnan K. Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks. In *Proc. the 33rd International Conference on Neural Information Processing Systems*, Dec. 2019, pp.4900–4909.

[53] Micikevicius P, Narang S, Alben J, Diamos G, Elsen E, Garcia D, Ginsburg B, Houston M, Kuchaiev O, Venkatesh G, Wu H. Mixed precision training. arXiv: 1710.03740, 2017. https://arxiv.org/abs/1710.03740#, April 2023.

[54] Hanson S J, Pratt L Y. Comparing biases for minimal network construction with back-propagation. In *Proc. the 1st International Conference on Neural Information Processing Systems*, Jan. 1988, pp.177–185.

[55] LeCun Y, Denker J S, Solla S A. Optimal brain damage. In *Proc. the Advances in Neural Information Processing Systems*, Nov. 1989. pp.598–605.

[56] Zhu M, Gupta S. To prune, or not to prune: Exploring the efficacy of pruning for model compression. In *Proc. the 6th International Conference on Learning Representations*, Apr. 2018.

[57] Han S, Pool J, Tran J, Dally W J. Learning both weights and connections for efficient neural networks. In *Proc. the 28th International Conference on Advances in Neural Information Processing Systems*, Dec. 2015. pp.1135–1143.

[58] Liu Z, Li J G, Shen Z Q, Huang G, Yan S M, Zhang C S. Learning efficient convolutional networks through network slimming. In *Proc. the 2017 IEEE International Conference on Computer Vision*, Oct. 2017, pp.2755–2763. DOI: 10.1109/ICCV.2017.298.

[59] Ye J B, Lu X, Lin Z, Wang J Z. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *Proc. the 6th International Conference on Learning Representations*, Apr. 2018.

[60] Fleischer B, Shukla S, Ziegler M, Silberman J, Oh J, Srinivasan V, Choi J, Mueller S, Agrawal A, Babinsky T, Cao M Z, Chen C Y, Chuang P, Fox T, Gristede G, Guillorn M, Haynie H, Klaiber M, Lee D, LO S H, Maier G,

Scheuermann M, Venkataramani S, Vezyrtzis C, Wang N G, Yee F, Zhou C, Lu P F, Curran B, Chang L, Gopalakrishnan K. A scalable multi-TeraOPS deep learning processor core for AI Trainina and inference. In *Proc. the 2018 IEEE Symposium on VLSI Circuits*, Jun. 2018, pp.35–36. DOI: 10.1109/VLSIC.2018.8502276.

[61] Li H, Pang Y R, Zhang J L. Security enhancements for approximate machine learning. In *Proc. the on Great Lakes Symposium on VLSI 2021*, Jun. 2021, pp.461–466. DOI: 10.1145/3453688.3461753.

[62] Leipnitz M T, Nazar G L. High-level synthesis of resource-oriented approximate designs for FPGAs. In *Proc. the 56th ACM/IEEE Design Automation Conference (DAC)*, Jun. 2019.

Hao-Hua Que is currently pursuing his B.S. degree in Beijing Forestry University, Beijing. His research interest includes artificial intelligence applications and machine learning algorithms based on approximate circuits.

Yu Jin is currently an undergraduate of Beijing Forestry University, majoring in electronic information science and technology, Beijing. His research interests include approximate computing in algorithm, and FPGA.

Tong Wang is currently an undergraduate from Beijing Forestry University, majoring in electronic information science and technology, Beijing. His research interests include digit circuit and approximate calculation.

Ming-Kai Liu is currently an undergraduate from Beijing Forestry University, Beijing, majoring in electronic information science and technology. His research interests include approximate computing in algorithm, signal analyzing and processing.

Xing-Hua Yang received his Ph.D. degree in electronic science and technology from Department of Electronics Engineering, Tsinghua University, Beijing, in 2017. He is currently working as a lecturer of College of Science, Beijing Forestry University, Beijing. His research interests include approximate computing in algorithm, circuit, and system design.

Fei Qiao received his Bachelor's degree in electronics and information system from Lanzhou University, Lanzhou, in 2000, and his Ph.D. degree in electronics science and technology from Tsinghua University, Beijing, in 2006. He is currently an associate professor with the Department of Electronic Engineering, Tsinghua University, Beijing. He has authored around 90 papers and holds over 30 invented patents. His research interests include low power circuits design, and energy-efficient integrated perception circuits and systems for intelligent robots, wearables and IoT devices.