AxPPLAs: Approximate Ling Adders Using Parallel Prefix Topologies

Yongqiang Zhang[1] | Qiwei Zhou[1] | Jie Han[2] | Guangjun Xie [1] | Xin Cheng [1]

[1]School of Microelectronics, Hefei University of Technology, Hefei, China
[2]Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Alberta, Canada

Corresponding
Xin Cheng, School of Microelectronics, Hefei University of Technology, Hefei, China.
Email: xcheng@hfut.edu.cn

Funding information
This work was supported in part by the National Natural Science Foundation of China under Grant 62404067, in part by the Fundamental Research Funds for the Central Universities under Grant JZ2025HGTB0231 and PA2025GDSK0034, and in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada under Grant RES0048688.

Abstract

While parallel prefix adders (PPAs) optimize data path through prefix operators, Ling adders show exceptional computing performance. Compared to traditional adders, however, Ling adders require additional components for generating Ling carries, so they are more complex. This paper proposes an architecture for approximate Ling adders based on parallel prefix topologies, referred to as approximate parallel prefix Ling adders (AxPPLAs). It is realized by using approximate prefix operators (AxPOs) to simplify the exact function of Ling carries for some less significant bits. The more significant bits are accurately computed. Various parallel prefix topologies and approximate processing bits are implemented to balance hardware costs and computing accuracy. The proposed AxPPLAs are compared with state-of-the-art designs in terms of hardware cost, accuracy, and other figures of merits. Experimental results show that the proposed 16-bit AxPPLAs improve hardware efficiency by significantly reducing delay and energy by up to 38.51% and 19.46%, on average, respectively, while maintaining competitive accuracy.

KEYWORDS

Approximate computing, Ling adder, parallel prefix topology, delay, energy efficiency.

1 | INTRODUCTION

Approximate computing (AxC) is considered promising to save area and power through a slight loss of computing accuracy[1]. Applications such as image, video and audio processing, wireless communication, data mining, and identification are inherently fault-tolerant and have the ability to withstand small errors in computing results. Approximate adders (AxAs) are one of the fundamental components in building inexact systems for these applications[2,3,4]. In addition, AxAs can also be used in more complex arithmetic units to achieve cross-level approximation, such as multipliers[5,6,7], dividers[8,9,10], and squarers[11,12,13].

Previous studies have proposed various AxAs, such as approximate parallel prefix adders (AxPPAs)[14], lower-part OR adders (LOAs)[15], error-tolerant adders (ETAs)[16], error reduced carry prediction approximate adders (ERCPAAs)[17], dual sub-adder based low latency adder (DSLLA)[2], low error and area efficient approximate adder (LEADx)[18], area and power efficient approximate adder (APEx)[18], approximate mirror adder 5 (AMA5)[19], and variable latency parallel prefix adders (VLPPAs)[20,21]. AxPPAs approximate results for less significant bits by simplifying prefix operations to reduce the number of logic gates required for generating carry-outs[14]. LOAs approximate several less significant bits using OR gates to reduce power consumption[15]. ETAs are a type of fault-tolerant adders that incorporate an error-correcting XOR. If an error detection signal in the *i*th bit is active, *i*-1 less significant bits are set to 1 to alleviate the strict precision constraints and save energy[16]. ERCPAAs utilize a carry prediction signal to reduce errors in approximated bits by setting them to a constant, such as 0 or 1[17]. DSLLA splits an accurate adder into two parts to reduce delay. Its approximate part supports various approximation methods. An error recovery technique is used to check if there is a difference between the carry predicted by a lower significance adder (LSA) and the actual carry. If there is a difference, different actions are taken based on the least significant bit (LSB) of an upper significance adder (USA). When the LSB is 0, the carry of LSA is combined with the LSB of USA through an OR operation to

correct errors; when the LSB is 1, all output bits of LSA are set to 1 to make results closer to the correct values[2]. LEADx is a low-error and area efficient approximate adder. In its approximate part with K bits, the two most significant sums are computed by an AAd1, and the remaining bits are computed by (K-2)/2 AAd2s. The carry of the exact part is obtained from AAd1. AAd1 adopts a high-precision approximate addition. AAd2 works as follows: set the carry C_{out} of the i-th group as the input A_{i+1} of this group. Compute S_i and S_{i+1} according to the predicted C_{out}. If the prediction is correct, perform an accurate output using the standard 2-bit addition. If the prediction is wrong and the predicted value of Cout is 0, set both sum outputs to 1. If the prediction is wrong and the predicted value of C_{out} is 1, set both sum outputs to 0. To further reduce area and power consumption, APEx simplifies the approximate logic based on the design of LEADx. In the K-bit approximate unit of APEx, the two most significant sums are accurately computed by the AAd1 module, while the remaining bits are directly set to logic 1[18]. AMA5 achieves a low-power design by using a mirroring operation (i.e., copying inputs directly to outputs) in its approximate part, setting $S_i=B_i$ and $C_{out}=A_i$[19]. VLPPAs generate correct results for less significant bits while speculating results for more significant bits through an error detection network[20,21]. All these designs aim to balance hardware costs and computing accuracy to meet the requirements of various fault-tolerant applications.

To speed up addition for saving energy, Ling adders generate Ling carries propagated from less significant bits to more significant bits, instead of the carry-outs in the aforementioned adders[23]. Although more components are required to generate Ling carries than those for carry-outs, a well-designed approximation can optimize area and power. Inspired by these motivations, we propose an approximate architecture for Ling adders using parallel prefix topologies (AxPPLAs). To this end, approximate prefix operators (AxPOs) are defined to optimize the exact logic expression for generating Ling carries for a part of less significant bits in AxPPLAs, while more significant bits are accurately computed. The key idea behind this method is the shorter critical path of the AxPOs compared to that of the exact one. This will trade some accuracy for improvements in terms of area, power, and energy. Six typical parallel prefix topologies, including Brent Kung[24] (AxPPLABK), Kogge Stone[25] (AxPPLAKS), Beaumont Smith[26] (AxPPLABS), Han Carlson[27] (AxPPLAHC), Knowles[28] (AxPPLAKN), and Sklansky[29] (AxPPLASK), are implemented for different approximated less significant bits to study the balance between speed and accuracy. Experimental results show that the proposed 16-bit AxPPLAs reduce delay, energy, power-delay product (PDP)-normalized mean error distance (NMED), and energy-delay product (EDP)-NMED by up to 38.51%, 19.46%, 37.35%, and 59.27% on average compared to previous designs with the same number of approximated bits.

The main contributions of this work are as follows:

1) An architecture for approximate parallel prefix Ling adders (AxPPLAs) that achieves a balance between hardware efficiency and computing accuracy.

2) AxPOs to synthesize less significant bits in AxPPLAs and to speed up operation and reduce energy.

3) The implementation of various parallel prefix topologies, including Brent Kung, Kogge Stone, Beaumont Smith, Han Carlson, Knowles, and Sklansky.

4) The evaluation of different approximated less significant bits in these designs in terms of error metrics, energy consumption, and other figures of metric (FoMs).

This paper proceeds as follows. Section 2 briefly reviews related works. Section 3 provides the basics of parallel prefix Ling adders. Section 4 illustrates the proposed approximate prefix operator and architecture for approximate parallel prefix Ling adders. Section 5 shows the experimental results. Section 6 concludes this work.

2 | RELATED WORKS

AxAs have garnered significant attention due to their potentials to save area, delay, and energy[30,31,32]. These architectures and techniques are briefly reviewed as follows.

2.1 | Reduced Delay/Critical Path

Several AxAs have been designed to shorten critical path to reduce delay and energy. Techniques such as gracefully degrading adders (GDAs)[33] and generic accuracy configurable adders (GeArs)[34] overlap multiple sub-adders to minimize critical path.

2.2 | Reduced Energy Consumption

Some AxAs aim to eliminate or simplify the generation of carry-outs to reduce power consumption. Examples include copy AxAs[35], truncation AxAs (Trunc-0, Trunc-1)[36], lower-part OR adders (LOAs)[15], error reduced carry prediction approximate adders (ERCPAAs)[17], and error-tolerant adder I (ETAI)[16]. These methods approximate less significant bits by using techniques such as copying inputs to results, setting results to constant values, performing bitwise OR operations, utilizing constant truncation, and applying XOR for error correction.

2.3 | Accuracy-Configurable Approaches

There are also accuracy-configurable approximate adders (ACAs)[37] and variable latency carry selection adders (VLCSAs)[38]. These adders offer configurable error detection and correction to support both exact and approximate operations to balance accuracy and hardware costs.

2.4 | Speculative Adders

Speculative adders like variable latency parallel prefix adders (VLPPAs)[20,21] generate exact results for less significant bits and predict results for more significant bits. These adders include preprocessing, speculative prefix processing, postprocessing, error detection, and error correction stages to effectively manage errors.

Overall, AxAs focus on optimizing hardware costs by balancing the trade-offs between accuracy, speed, power, and area. These advancements are crucial for fault-tolerant applications ranging from embedded systems to high-performance computing, such as AI, machine learning, data mining, image processing, audio processing[22].

3 | BASICS

Parallel prefix adders (PPAs) have been extensively studied in optimizing data paths for speeding up addition[14]. Ling adders are a type of faster adders because fewer steps are required[23]. Ling adders using parallel prefix topologies (PPLAs) compute in three stages: preprocessing, prefix computing, and postprocessing. Unlike PPAs, the generate g_i, propagate p_i, and half sum d_i of each bit in the preprocessing stage of an n-bit PPLA are defined as

$$g_i = a_i \cdot b_i, \tag{1}$$

$$p_i = a_i + b_i, \tag{2}$$

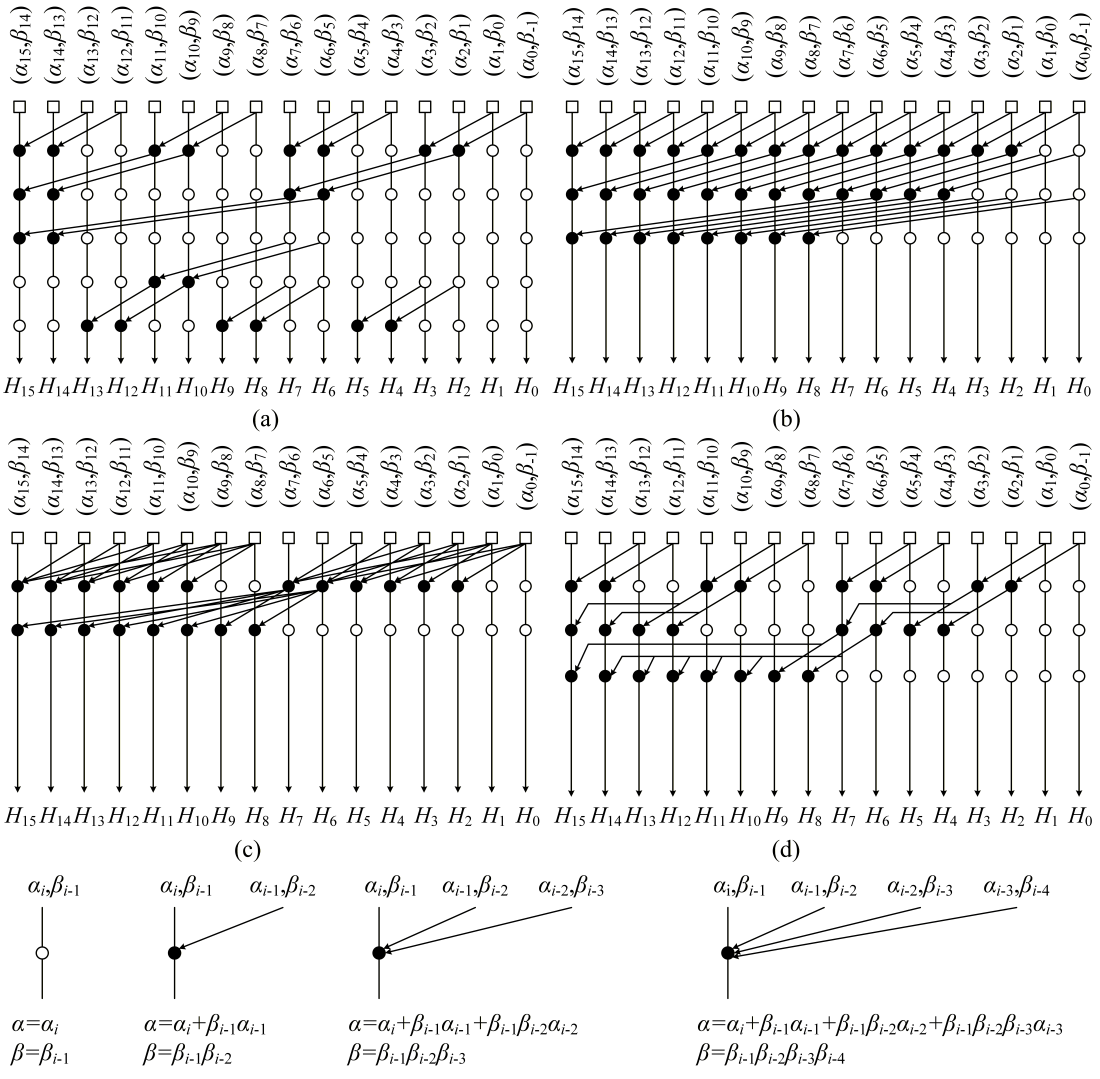

FIGURE 1 Examples of prefix processing stages using different parallel prefix topologies (a) Brent Kung, (b) Kogge Stone, (c) Beaumont Smith, (d) Sklansky, and (e) the generation of α and β.

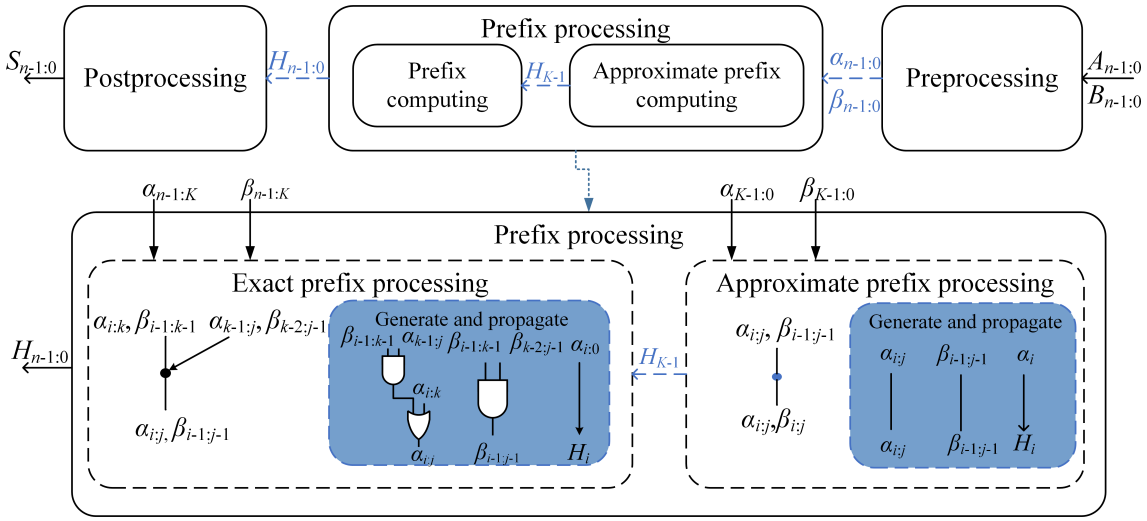

FIGURE 2 The proposed architecture for AxPPLAs.

$$d_i = a_i \oplus b_i, \tag{3}$$

where $0 \le i \le n-1$, the symbols $\cdot$, $+$, and $\oplus$ denote AND, OR, and XOR operations respectively, and the symbol $\cdot$ is omitted later. The carry-out c_i is then computed as

$$c_i = g_i + p_i \cdot c_{i-1}. \tag{4}$$

Ling has devised a carry generation method, denoted as Ling carry H as[23]

$$H_i = c_i + c_{i-1} \tag{5}$$

Thus, it can be denoted as a recursive expression using (4) as

$$H_i = g_i + g_{i-1} + p_{i-1}g_{i-2} + p_{i-1}p_{i-2}g_{i-3} + \cdots + p_{i-1}p_{i-2}\cdots p_2 p_1 g_0. \tag{6}$$

To simplify (6), two intermediate signals α_i and β_i for each bit in the preprocessing stage are defined as

$$\alpha_i = g_i + g_{i-1}, \tag{7}$$

$$\beta_i = p_i \cdot p_{i-1}, \tag{8}$$

where g_i, p_i, g_{i-1}, and p_{i-1} are taken from the preprocessing stage and $g_{-1}=0$ and $p_{-1}=0$. They are combined in the prefix processing stage to compute Ling carries according to various parallel prefix topologies. Taking (7) and (8) into (6), Ling carries can be respectively divided into two expressions for even and odd bits as H_{2k} and H_{2k+1}

$$H_{2k} = \alpha_{2k} + \beta_{2k-1}\alpha_{2k-2} + \beta_{2k-1}\beta_{2k-3}\alpha_{2k-2}\alpha_{2k-4} + \cdots + \beta_{2k-1}\beta_{2k-3}\cdots\beta_1\alpha_0, \tag{9}$$

$$H_{2k+1} = \alpha_{2k+1} + \beta_{2k}\alpha_{2k-1} + \beta_{2k}\beta_{2k-2}\alpha_{2k-1}\alpha_{2k-3} + \cdots + \beta_{2k}\beta_{2k-2}\cdots\beta_2\alpha_1, \tag{10}$$

where $0 \le k \le n/2-1$. To simplify the expressions of Ling carries, several definitions are given as

$$\alpha_{[2k:2(k-j)]} = \begin{cases} \alpha_{2k} & j=0 \\ \alpha_{[2k:2(k-m)]} + \beta_{[2k-1:2(k-m)-1]}\alpha_{[2(k-m-1):2(k-j)]} & else \end{cases},$$
$$\beta_{[2k:2(k-j)]} = \begin{cases} \beta_{2k} & j=0 \\ \beta_{[2k:2(k-m)]}\beta_{[2(k-m-1):2(k-j)]} & else \end{cases} \tag{11}$$

$$\alpha_{[2k+1:2(k-j)+1]} = \begin{cases} \alpha_{2k+1} & j=0 \\ \alpha_{[2k+1:2(k-m)+1]} + \beta_{[2k:2(k-m)]}\alpha_{[2(k-m)-1:2(k-j)+1]} & else \end{cases}$$
$$\beta_{[2k+1:2(k-j)+1]} = \begin{cases} \beta_{2k+1} & j=0 \\ \beta_{[2k+1:2(k-m)+1]}\beta_{[2(k-m)-1:2(k-j)+1]} & else \end{cases} \tag{12}$$

where $0 \le m < j \le k \le n/2-1$. The expressions for Ling carries H_{2k} and H_{2k+1} for even and odd bits can be written as

$$H_{2k} = \alpha_{[2k:0]}, \tag{13}$$

$$H_{2k+1} = \alpha_{[2k+1:1]}. \tag{14}$$

To clarify the Ling carry formulations in (9)-(14), we expand two example carries. For an even bit position, the Ling carry H_4 can be written as $H_4=\alpha_{[4:0]}=\alpha_4+\beta_3\cdot\alpha_2+\beta_3\cdot\beta_1\cdot\alpha_0$. For an odd bit position, the Ling carry H_5 can be written as $H_5=\alpha_{[5:1]}=\alpha_5+\beta_4\cdot\alpha_3+\beta_4\cdot\beta_2\cdot\alpha_1$.

The postprocessing stage computes the sum and carry-out as

$$c_{n-1} = p_{n-1} \cdot H_{n-1}, \tag{15}$$

$$s_i = d_i \oplus (p_{i-1} \cdot H_{i-1}), \tag{16}$$

TABLE 1 Truth table of the exact POs and proposed AxPOs

A_i	B_i	A_{i-1}	B_{i-1}	C_{in}	Exact			Approximate		
					c_i	c_{i-1}	S_i	c_i	c_{i-1}	S_i
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0
0	0	0	1	1	0	1	1	0	1	1
0	0	1	0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	1	0	1	1
0	0	1	1	0	0	1	1	0	1	1
0	0	1	1	1	0	1	1	0	1	1
0	1	0	0	0	0	0	1	0	0	1
0	1	0	0	1	0	0	1	0	0	1
0	1	0	1	0	0	0	1	0	0	1
0	1	0	1	1	1	1	0	0 (✗)	1	0
0	1	1	0	0	0	0	1	0	0	1
0	1	1	0	1	1	1	0	0 (✗)	1	0
0	1	1	1	0	1	1	0	1	1	0
0	1	1	1	1	1	1	0	1	1	0
1	0	0	0	0	0	0	1	0	0	1
1	0	0	0	1	0	0	1	0	0	1
1	0	0	1	0	0	0	1	0	0	1
1	0	0	1	1	1	1	0	0 (✗)	1	0
1	0	1	0	0	0	0	1	0	0	1
1	0	1	0	1	1	1	0	0 (✗)	1	0
1	0	1	1	0	1	1	0	1	1	0
1	0	1	1	1	1	1	0	1	1	0
1	1	0	0	0	1	0	0	1	0	0
1	1	0	0	1	1	0	0	1	0	0
1	1	0	1	0	1	0	0	1	0	0
1	1	0	1	1	1	1	1	1	1	1
1	1	1	0	0	1	0	0	1	0	0
1	1	1	0	1	1	1	1	1	0 (✗)	0 (✗)
1	1	1	1	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

where $s_0=d_0 \oplus c_{-1}$ is computed as a special case. (16) shows that the sum is computed by parallel bitwise XOR operations between c_{i-1} and d_i for all bits.

FIGURE 1 shows four prefix processing architectures for PPLAs: 1) Brent Kung, 2) Kogge Stone, 3) Beaumont Smith, and 4) Sklansky. A PPLA divides a prefix processing stage into odd and even bits to reduce logic levels and accelerate computing speed. A Brent Kung adder uses grouped 2 bits to compute prefixes for 4-bit groups, then for 8-bit groups iteratively. This topology undergoes $2\log2(n)$-3 logic levels, with less fan-in, fan-out, and wiring congestion, as shown in FIGURE 1(a). A Kogge Stone adder can independently compute each bit in a regular grid for efficiency and reducing the number of fan-outs, as shown in FIGURE 1(b). A Beaumont Smith adder is based on a low logic depth design with multiple fan-ins to reduce critical path or interconnect lengths, as shown in FIGURE 1(c). Sklansky adder, also known as a divide-and-conquer adder, optimizes the delay of intermediate prefix processing stages. However, this design doubles the number of fan-outs at each stage, which will lead to severe fan-out problems, as shown in FIGURE 1(d). FIGURE 1(e) gives the generation of intermediate signals α and β with one, two, three, and four inputs and the corresponding logic expressions are provided as well for them.

4 | THE PROPOSED DESIGNS

4.1 | Overall Architecture

The proposed architecture for n-bit AxPPLAs is shown in FIGURE 2. The overall architecture consists of three stages: preprocessing for generating signals α and β, prefix processing for computing Ling carries, and postprocessing for outputting results. In the preprocessing stage, two n-bit binary numbers A and B are processed to generate intermediate signals α and β for all bits. This stage uses exact logic to provide reliable inputs for the next stage. Unlike the exact PPLAs, the architecture partitions the prefix processing stage into an exact unit and an approximate one using a parameter K. The n-K more significant bits are computed with exact logic to ensure the required accuracy, while the K less significant bits are approximated to improve hardware efficiency. These two units are connected through a Ling carry H_{K-1} or a carry-out c_{K-1} propagated from the approximate unit to the exact one. The exact prefix processing unit generates accurate Ling carries H_{2k} and H_{2k+1}, while the approximate Ling carries H_{2k} and H_{2k+1} in the approximate unit are realized through approximate prefix operators (AxPOs) for even and odd bits, respectively as follows

$$H_{2k} \approx \alpha_{2k}, \tag{17}$$

$$H_{2k+1} \approx \alpha_{2k+1}. \tag{18}$$

The rationale behind the proposed AxPOs definitions is explained as follows. The approximation strategy adopted in this paper is not derived from arbitrary logic simplification, but directly deduced from the structural characteristics of the Ling

Algorithm 1 Pseudo code for *K*-bit approximation of *n*-bit AxPPLAs

```
Input: A, B, n, K
Output S
  for i=0 to n-1 do
      p(i) ← A(i) OR B(i)
      g(i) ← A(i) AND B(i)
      d(i) ← A(i) XOR B(i)
  end for
if K>0 then [Approximate logic]
  α(1,0) ← g(1,0)
  β(1,0) ← 0
  S(0) ← d(1,0)
  for i=1 to K-1 do
      α(1,i) ← g(i) OR g(i-1)
      β(1,i) ← p(i) AND p(i-1)
      S(i) ← d(i) XOR (α(1,i-1) AND p(1,i))
  end for
  H(K-1) ← α(1,K-1)
end if
if (n-K) > 0 then [Exact logic]
  α(1,K) ← g(1,K)
  β(1,K) ← d(1,K)
  for i=(K+1) to n-1 do
  for i=(K+1) to n-1 do
      α(1,i) ← g(i) OR g(i-1)
      β(1,i) ← p(i) AND p(i-1)
  end for
  y ← 1
  for i=1 to ( L ← logic levels in Fig. 1) do
   for j=(K+y+1)/2 to (n/2-1) do
    α(i+1,2j) ← α(i,2j) OR ( α(i,2j-y) AND
    β(i,2j-1))
    β(i+1,2j-1) ← β(i,2j-1) AND β(i,2j-1-y)
    α(i+1,2j+1) ← α(i,2j+1) OR
                  (α(i,2j+1-y) AND β(i,2j))
    β(i+1,2j) ← β(i,2j) AND β(i,2j-y)
     end for
   y ← 2i-1
  end for
  cin ← H(K-1)
  for i=K+2 to (n-1) do
     H(i) ← α(L+1, i) OR ( cin AND β(L+1,i-1))
     S(i) ← d(i) XOR (H(i) AND p(1,i))
  end for
  S(n) ← H(n-1) AND p(1,n-1)
end if
```

adder. The proposed AxPOs simplify the generation of Ling carries in the approximated region by directly forwarding a single signal. Specifically, this wire-level approximation completely eliminates the AND and OR gates required in the exact post-processing logic. Thus, the critical path is significantly shortened, while the logic fan-in and routing complexity are also reduced accordingly. These advantages are particularly pronounced in grid-based parallel-prefix structures such as Kogge–Stone and Han–Carlson.

From the perspective of accuracy, selecting α as the approximation target is also well justified. According to Equation (7), $\alpha_i=g_i+g_{i-1}$ represents a generate-dominant condition. This signal can capture the strongest evidence of carry generation among adjacent bits. In contrast, $\beta_i=p_i \cdot p_{i-1}$ only describes the propagation behavior. If β is retained in the absence of a generate event, it may lead to spurious carry propagation and introduce systematic bias. Thus, approximating H with α can be regarded as a conservative replacement for the full carry logic. In this case, errors occur primarily in input patterns that rely on long propagation chains. These chains are intentionally omitted in the approximation process. The limited error patterns and their probability analysis results presented in Table 1 are consistent with the design expectations.

It should be further noted that although the exact Ling carry can be expanded as (9) and (10), the design objective of AxPOs is not to truncate this numerical expression term by term. Its core goal is to structurally eliminate the cross-bit propagation dependency of carries. If any β-related term is retained in the approximation (e.g., $H_i \approx \alpha_i+\beta_{i-1}$ or $H_i \approx \alpha_i+\beta_{i-1} \cdot \alpha_{i-1}$), it will inevitably require accessing the intermediate signals of adjacent or lower-order bits. This will reintroduce recursive dependencies and prefix computation logic, while undermining the advantages of AxPOs in terms of critical path and routing complexity. In contrast, retaining only α can completely eliminate cross-bit propagation dependencies while preserving the core semantics of carry generation. Since α represents a generate-dominant condition whereas β merely reflects the propagation behavior, the proposed AxPOs can be regarded as the only rational, low-risk approximation point naturally derived from the Ling carry structure, among all candidate approximations that simultaneously maintain locality and non-recursiveness. This scheme achieves a favorable balance between critical path optimization and error controllability, which is also supported by the theoretical error enumeration analysis and experimental results presented in this paper.

In the postprocessing stage, the addition is completed using Ling carries generated in the prefix processing stage and the propagate signals from the preprocessing stage through (15) and (16). This stage also uses exact logic to ensure the accuracy of final results.

In theory, the i-th bit carry within the approximation unit only depends on the inputs of i-th and (i-1)-th bits. This reduces the complexity of a carry generation. Table 1 shows the truth table of i-th bit carry and sum generated through exact POs and the proposed AxPOs, where error cases are marked with a cross. This results in an error probability of about 3.125% for sum (S_i) and a 12.5% error probability for carry (c_i).

This analysis means that the AxPOs are used instead of exact Ling carries to approximate results by directly hardwiring the outputs from the preprocessing stage to the inputs of the postprocessing stage. This can also be seen in FIGURE 2. Thus, the approximate Ling carries H_{2k} and H_{2k+1} are only related to the current intermediate signals through the proposed AxPOs and neglect them coming from less significant bits, which will cause some computing accuracy loss. For verifying the efficacy of this method, six parallel prefix topologies including Brent Kung ($AxPPLA_{BK}$), Kogge Stone ($AxPPLA_{KS}$), Beaumont Smith ($AxPPLA_{BS}$), Han Carlson ($AxPPLA_{HC}$), Knowles ($AxPPLA_{KN}$), and Sklansky ($AxPPLA_{SK}$) will be implemented for different approximated bits to study the tradeoff between hardware costs and computing accuracy.

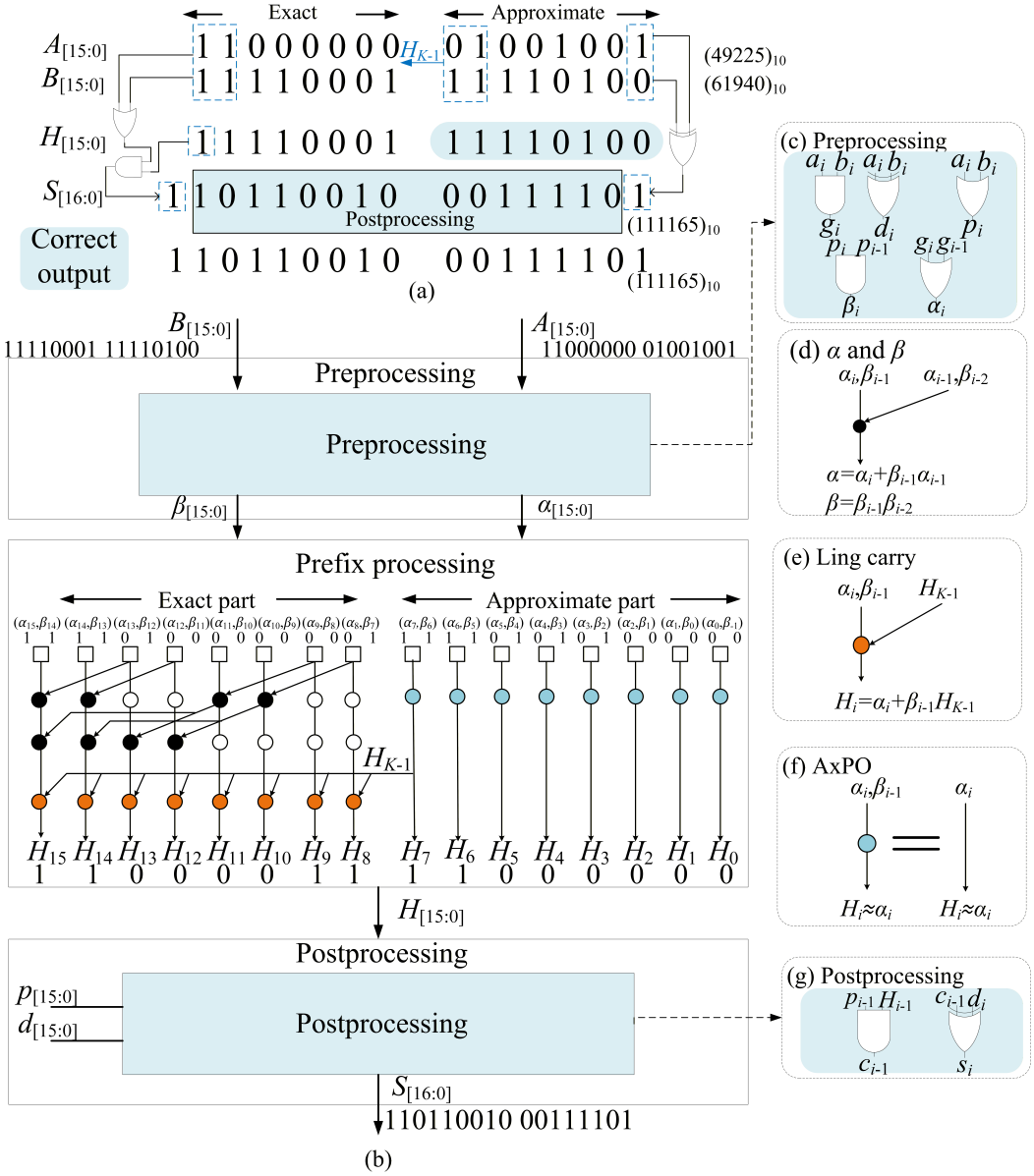

FIGURE 3 An example for the proposed AxPPLAs. (a) Inputs and outputs. (b) Sklansky topology for n=16 and K=8. (c) Preprocessing. (d) The generation of intermediate signals α and β. (e) Ling carry. (f) AxPO. (g) Postprocessing.

4.2 | Operations of the Proposed AxPPLAs

FIGURE 3 shows an example for illustrating the working principle of the AxPPLAs according to the proposed architecture. The Sklansky topology is employed; the adder size and approximated bits are set to n=16 and K=8.

In FIGURE 3(a), the exact sum of 16-bit input operands $A_{[15:0]}=a_{15}a_{14}\ldots a_0=(49225)_{10}$ and $B_{[15:0]}=b_{15}b_{14}\ldots b_0=(61940)_{10}$ is $S_{[16:0]}=s_{16}s_{15}\ldots s_0=(111165)_{10}$, where subscript 10 is the radix. FIGURE 3(b) shows the circuit of AxPPLA$_{SK}$, where the preprocessing stage generates α, β, and d for each bit of operands according to (7), (8), and (3). The employed components in the preprocessing stage consist of simple AND, OR, and XOR gates, as shown in FIGURE 3(c). The approximate prefix processing unit passes the value of α in the preprocessing stage to the postprocessing stage through wires marked with light green cycles, approximating H_0~H_7 as α_0~α_7 using the AxPOs in FIGURE 3(f), for K=8 less significant bits. The AxPOs implement the structures defined in (17) and (18) to optimize the expressions of Ling carries by eliminating both an OR gate and an AND gate. Meanwhile, the approximate unit provides a one-bit Ling carry H_7 for the exact unit, marked with orange cycles. FIGURE 3(d) and FIGURE 3(e) are used in the exact unit for generating n-K=16-8=8 more significant Ling carries H_8~H_{15} using (13) and (14). The final sum is computed in the postprocessing stage by AND and XOR operations in FIGURE 3(g), according to (15) and (16). With these steps, the sum bits s_0~s_7 are approximated, and bits s_8~s_{16} are exactly computed. Based on these Ling carries, the AxPPLA$_{SK}$ yields the same approximate sum.

Algorithm 1 represents an n-bit AxPPLA with K approximate bits. It divides n bits into two independent units: a K-bit approximate logic (lines 8-18) and an n-K-bit exact logic (lines 29-42). This significantly reduces latency and hardware costs while ensuring the usability of computing results. When i ranges from 0 to K-1, approximate logic computes the sum (lines 8-18), bypassing computing all prefix operators. The simplified intermediate signal α is approximated as Ling carry H. When i ranges from K to n-1, it switches to exact logic for an accurate sum (lines 19-42). In the prefix processing stage (lines 2-35),

TABLE 2 MRED, MED and NMED of various approximate adders

Parameters (n,K)	Design	MRED	MED	NMED	Parameters (n,K)	Design	MRED	MED	NMED
(16,2)	AxPPA	1.04e-05	4.91e-01	3.76e-06	(32,4)	AxPPA	1.13e-09	3.52e+00	4.10e-10
	AxPPLA	5.02e-06	2.50e-01	1.87e-06		AxPPLA	8.06e-10	2.50e+00	2.92e-10
	ERCPPA	1.59e-05	7.49e-01	5.72e-06		ERCPPA	1.52e-09	4.71e+00	5.49e-10
	LOA	1.33e-05	6.24e-01	4.77e-06		LOA	9.25e-10	2.86e+00	3.34e-10
	ETA	1.47e-05	6.85e-01	5.24e-06		ETA	1.01e-09	3.13e+00	3.66e-10
	DSLLA	6.98e-05	3.26e+00	2.49e-05		DSLLA	8.10e-09	2.46e+01	2.87e-09
	LEADx	6.66e-06	3.16e-01	2.41e-06		LEADx	2.21e-10	6.92e-01	8.08e-11
	APEx	2.95e-05	1.37e+00	1.05e-05		APEx	3.99e-10	1.25e+00	1.45e-10
	AMA5	2.13e-05	1.00e+00	7.67e-06		AMA5	1.28e-09	3.99e+00	4.65e-10
(16,4)	AxPPA	7.43e-05	3.48e+00	2.66e-05	(32,8)	AxPPA	2.03e-08	6.34e+01	7.40e-09
	AxPPLA	5.34e-05	2.48e+00	1.89e-05		AxPPLA	1.50e-08	4.68e+01	5.46e-09
	ERCPPA	5.53e-05	2.59e+00	1.98e-05		ERCPPA	9.69e-09	3.03e+01	3.53e-09
	LOA	6.13e-05	2.88e+00	2.20e-05		LOA	1.53e-08	4.79e+01	5.58e-09
	ETA	6.65e-05	3.14e+00	2.40e-05		ETA	1.64e-08	5.12e+01	5.97e-09
	DSLLA	1.72e-04	8.24e+00	6.30e-05		DSLLA	1.20e-07	3.79e+02	4.42e-08
	LEADx	1.49e-05	6.90e-01	5.27e-06		LEADx	3.71e-08	1.15e+02	1.34e-08
	APEx	2.68e-05	1.25e+00	9.55e-06		APEx	6.82e-09	2.13e+01	2.48e-09
	AMA5	8.47e-05	3.98e+00	3.04e-05		AMA5	2.05e-08	6.40e+01	7.47e-09
(16,8)	AxPPA	9.97e-04	4.71e+01	3.60e-04	(32,16)	AxPPA	5.26e-06	1.64e+04	1.91e-06
	AxPPLA	9.64e-04	4.66e+01	3.56e-04		AxPPLA	3.84e-06	1.20e+04	1.40e-06
	ERCPPA	6.50e-04	3.07e+01	2.35e-04		ERCPPA	2.43e-06	7.57e+03	8.83e-07
	LOA	1.01e-03	4.80e+01	3.67e-04		LOA	3.92e-06	1.23e+04	1.43e-06
	ETA	1.08e-03	5.11e+01	3.91e-04		ETA	4.17e-06	1.31e+04	1.53e-06
	DSLLA	1.44e-03	6.93e+01	5.30e-04		DSLLA	3.14e-05	1.02e+05	1.18e-05
	LEADx	2.70e-04	1.26e+01	9.61e-05		LEADx	1.04e-05	3.25e+04	3.79e-06
	APEx	4.53e-04	2.13e+01	1.63e-04		APEx	5.26e-06	1.64e+04	1.91e-06
	AMA5	1.36e-03	6.42e+01	4.91e-04		AMA5	5.26e-06	1.64e+04	1.91e-06

based on the applied parallel prefix tree structure, Ling carry H is computed iteratively using intermediate signals α and β. In the post-processing stage (lines 36-42), the sum S_i and the carry c_i are output in parallel. To verify the efficacy of the AxPPLA algorithm, the designed AxPPLAs are applied to the Gaussian smoothing algorithm for image processing as a typical case to conduct a systematic test and analysis.

5 | EXPERIMENTAL RESULTS

In this section, the performances of the proposed AxPPLAs are evaluated and compared with previous representative counterparts in terms of hardware costs and computing accuracy. In real applications, to control errors, the approximate bit number K of the 16-bit adder (n=16) is set to 2, 4, and 8, while that of the 32-bit adder (n=32) is set to 4, 8, and 16. The cases with excessively large K values that cause intolerable errors are excluded. All designs are programmed in Verilog hardware description language (HDL) and synthesized using Synopsys Design Compiler (DC) with a TSMC 40 nm CMOS standard library at the typical corner. The clock period is set to be 10 ns. The command 'compile_ultra' is used to ungroup all entities to automatically synthesize all adders. For fair comparison, all adders, including the proposed AxPPLAs and the reference designs, are synthesized under the same conditions described above.

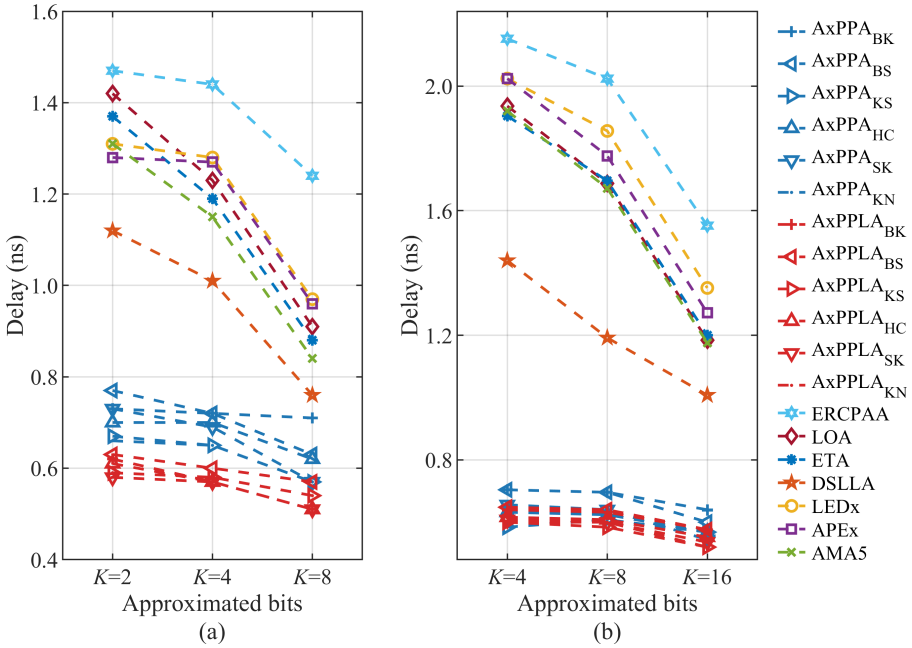

FIGURE 4 Delay of various approximate adders. (a) Dealy at n=16 bits. (b) Delay at n=32 bits.

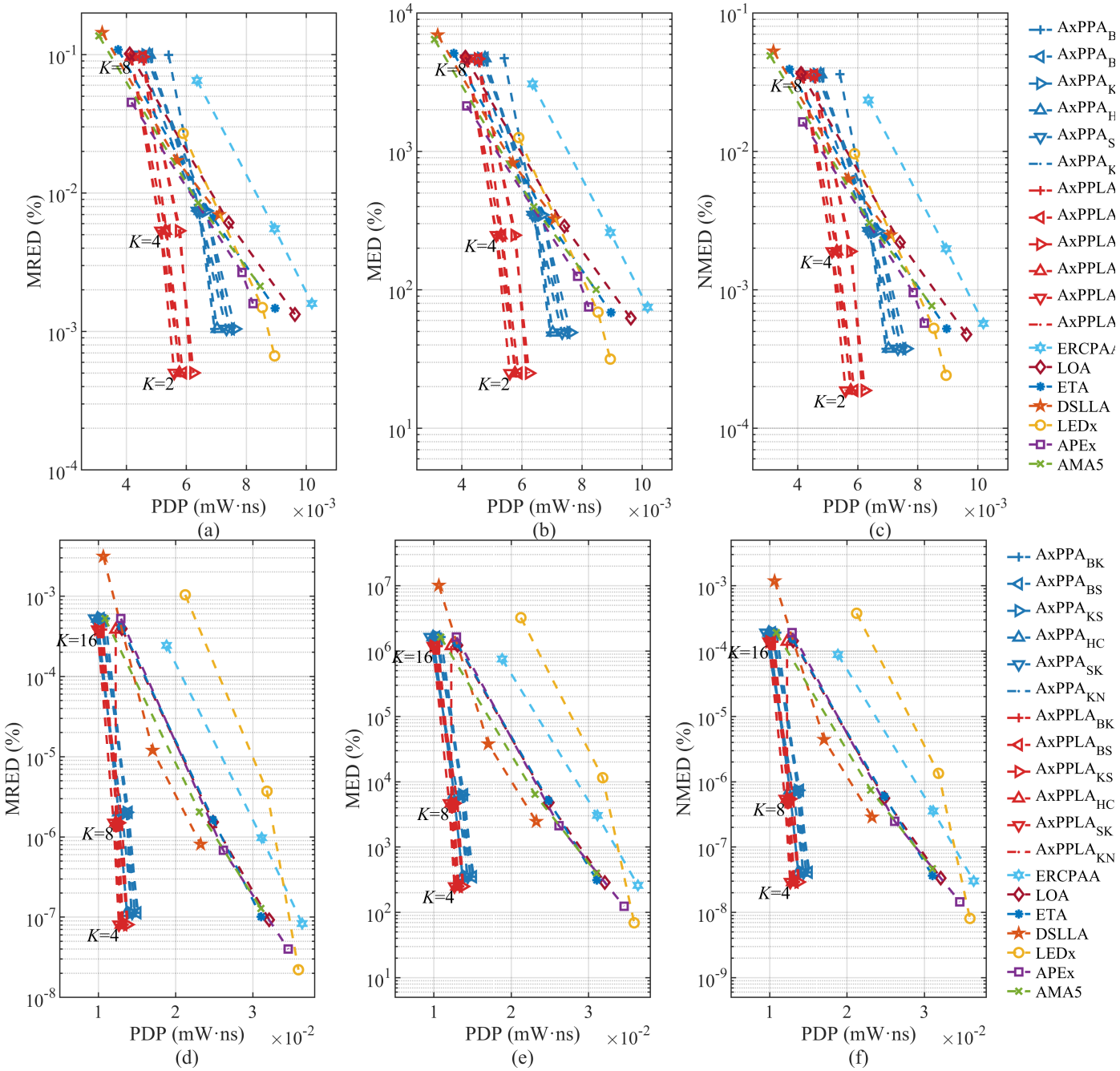

FIGURE 5 PDP versus MRED, MED, and NMED for various approximate adders. (a) MRED at n=16 bits. (b) MED at n=16 bits. (c) NMED at n=16 bits. (c) MRED at n=32 bits. (b) MED at n=32 bits. (c) NMED at n=32 bits.

A test set with 10^7 uniformly distributed random input pairs is produced in MATLAB for assessing the accuracy of considered adders. Error distance (ED) shows the arithmetic difference between approximate results M' and accurate results M and is computed as $ED=|M'-M|$[39] Relative error distance (RED) shows the difference relative to accurate results, computed as $RED=|ED/M|$. ED and RED reflect two key features of an approximate design. If two designs produce the same ED, the one with a larger RED is more accurate than the other one. Mean ED (MED) and mean relative ED (MRED) are the averages of all EDs and REDs and are computed as

$$MED = \sum_{i=1}^{N} ED_i \cdot P(ED_i), \tag{19}$$

$$MRED = \sum_{i=1}^{N} RED_i \cdot P(RED_i), \tag{20}$$

where N is the total number of input pairs, ED_i and RED_i are the ED and RED of the ith input, respectively. $P(ED_i)$ and $P(RED_i)$ are the probabilities that ED_i and RED_i occur. Normalized mean error distance (NMED) is defined as the normalization of MED by the maximum result S of an accurate design. It is used to compare the error magnitudes of approximate designs with different sizes and is computed as

$$NMED = MED / MAX(S). \tag{21}$$

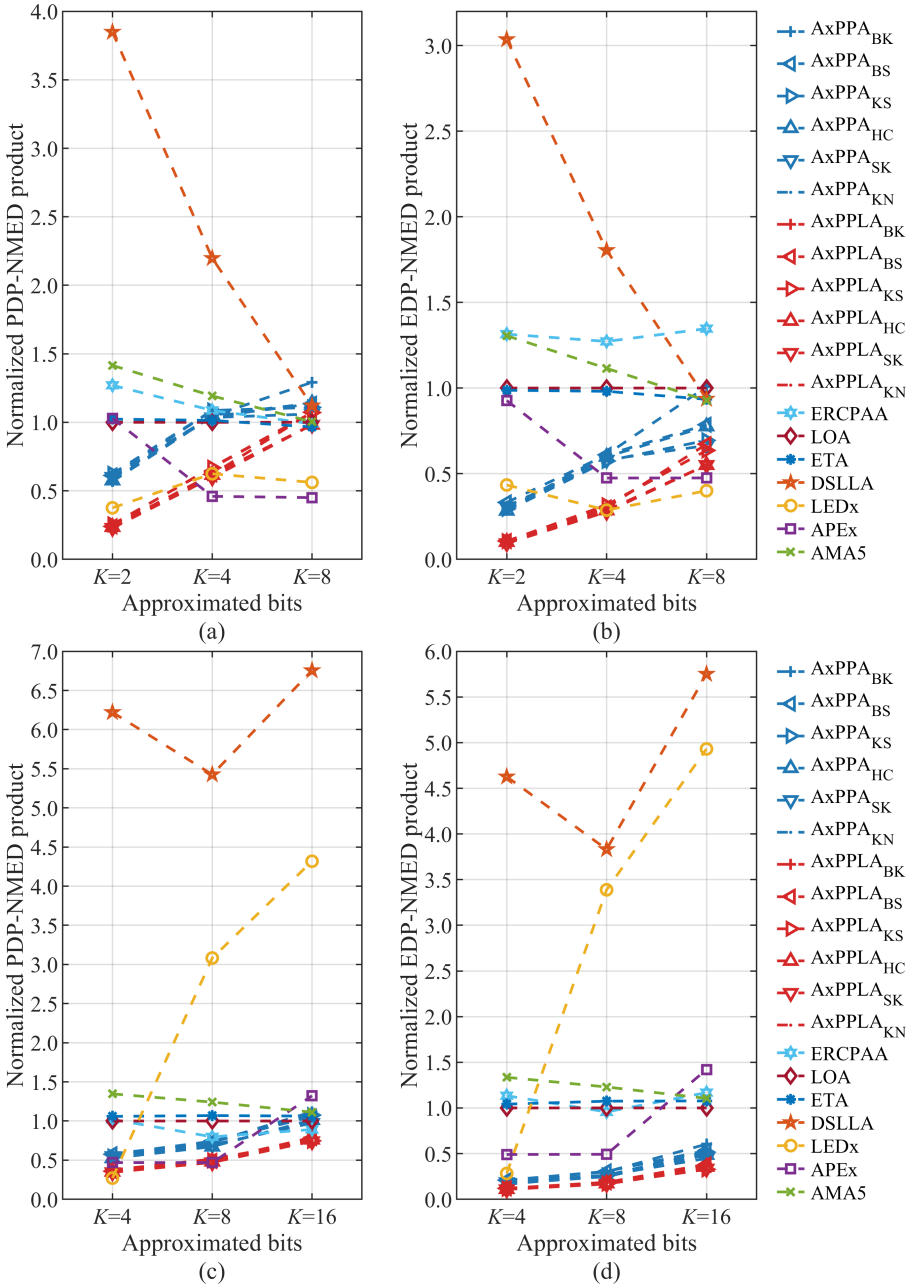

FIGURE 6 Normalized PDP-NMED products and EDP-NMED products for various approximate adders. (a) PDP-NMED at n=16 bits. (b) EDP-NMED at n=16 bits. (c) PDP-NMED at n=32 bits. (d) EDP-NMED at n=32 bits.

5.1 | Accuracy

Table 2 shows the MRED, MED, and NMED of considered AxAs with different approximated bits K. The data shows that the error performance of AxPPLAs significantly surpasses most designs. Its accuracy advantage stems from deep optimization of carry logic, rather than relying on additional error repair modules or dedicated low-error logic. For large values of K, the errors are slightly higher than those of low-error architectures using active prediction mechanisms (such as ERCPPAs), but these error-recovery designs have higher hardware costs. Take LEADx as an example. Its approximate unit controls errors through a high-precision unit (AAd1) for high-order bits and a grouped prediction unit (AAd2) for low-order bits, but the complex logic leads to increased latency. In contrast, AxPPLAs achieve low-order bit accuracy through global prefix logic optimization. For K=2, it only needs to handle carry approximation errors, thus significantly reducing overall errors. Compared with all previous AxAs, the MRED, MED, and NMED are reduced by 76.07%, 74.62%, and 75.09% for the case of K=2, respectively, with an average reduction of 31.05%, 30.25%, and 30.42% for three metrics for all values of K.

For 32-bit adders, AxPPLAs continues and strengthens this advantage. For K=2, the average reductions in MRED, MED, and NMED reach 53.26%, 53.46%, and 53.46%, and the error reduction remains above 50% as K increases. This indicates that AxPPLAs maintain stable and excellent accuracy performance regardless of bit width or approximation bits.

5.2 | Delay

FIGURE 4 shows the delay of these AxAs with different numbers of approximated bits. The proposed AxPPLAs

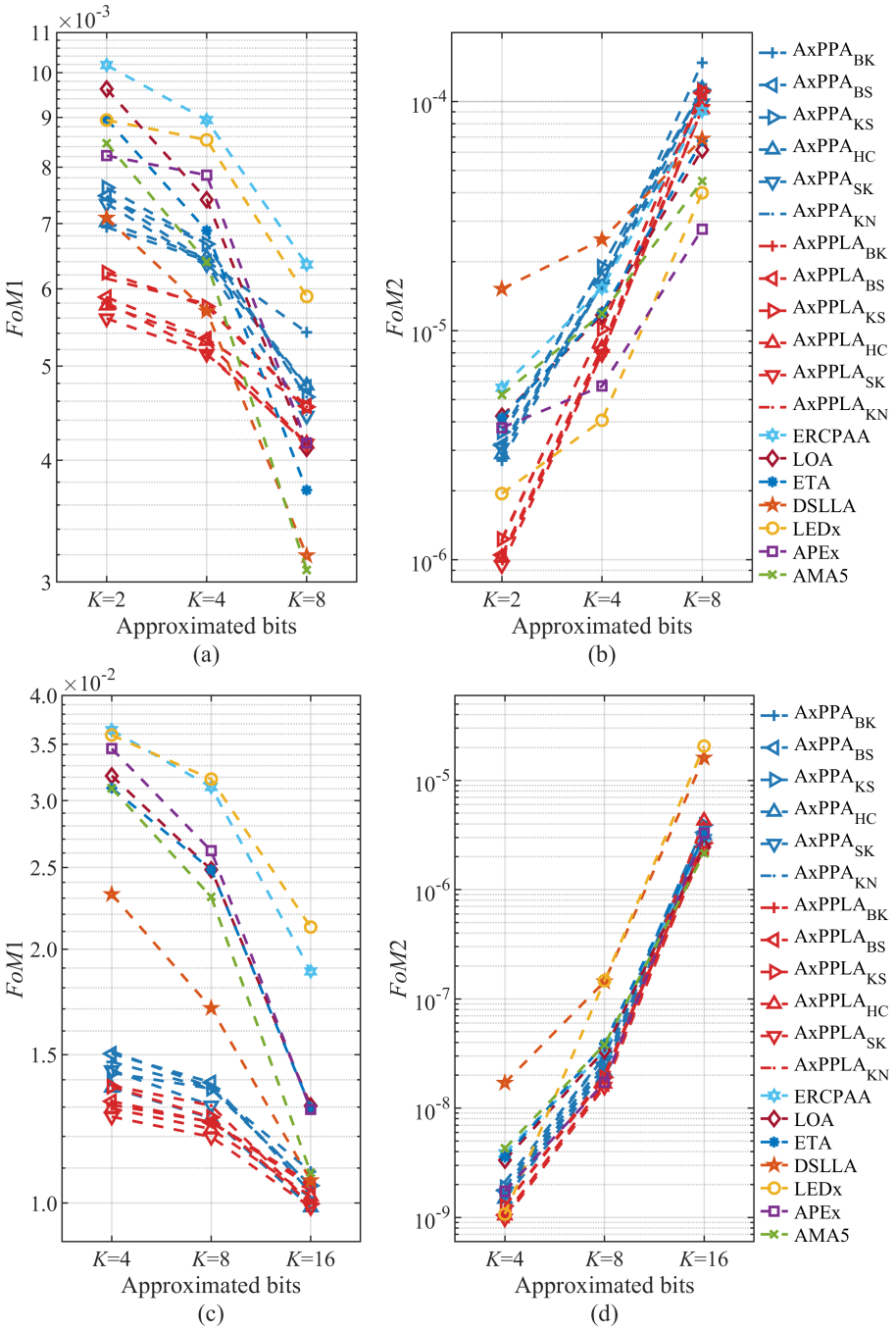

FIGURE 7 FoMs of various approximate adders. (a) FoM1 at n=16 bits. (b) FoM2 at n=16 bits. (c) FoM1 at n=32 bits. (d) FoM2 at n=32 bits.

consistently show smaller delays than AxPPAs with the same parallel prefix topology, for all values of K. Other AxAs like ETAs, ERCPAAs, and LOAs exhibit much larger delays across all values of K, ranging between approximately 0.8 ns and 1.5 ns in FIGURE 4(a). In detail, the proposed AxPPLAs reduce delay by up to 14.79% and 38.51% for three cases K=2, 4, and 8 on average, compared with AxPPAs and all mentioned AxAs, respectively. These data indicate that AxPPLAs achieve excellent performance in terms of computing speed, useful in accelerating addition in applications. In the case of 32-bit adders shown in FIGURE 4(b), AxPPLAs present the same advantage. The proposed AxPPLAs reduce delay by up to 59.65% for three cases K=4, 8, and 16 on average, compared with all mentioned AxAs, respectively.

5.3 | Computing Accuracy Versus PDP

Considering only computing accuracy for AxAs is biased, thus hardware costs are taken into consideration FIGURE 5 compares MRED, MED, and NMED with respect to PDP consumption for the AxAs when n=16, K=2, 4, and 8, as well as when n=32, K =4, 8, and 16. For different values of K, these AxAs show the same trends in the relationship between accuracy and PDP. For example, as the value of K increases, energy and accuracy decrease at the same time. LOAs use OR gates to approximate less significant bits, while ETAs incorporate an error-correcting XOR gate to set less significant bits to 1 if an error detection signal is active, and AMA5s copy inputs directly to outputs, resulting in slightly lower energy consumption than AxPPLAs at K=8, but with noticeably lower accuracy. Although ERCPAAs offer better accuracy for K=8, their high energy consumption is a significant drawback. For APEXs, the two MSBs of the approximate unit are accurately computed by the low-error module, while the remaining bits are directly set to logic 1. This makes its performance slightly better than AxPPLAs

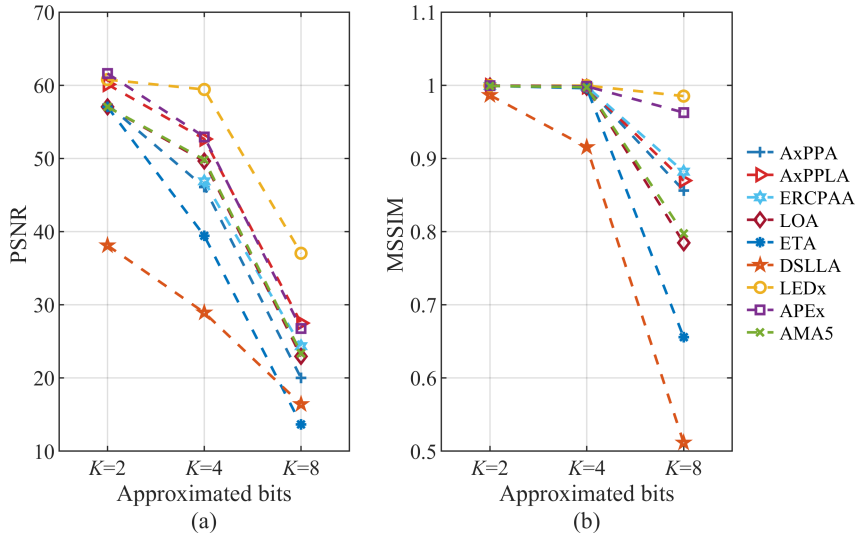

FIGURE 8 PSNR and MSSIM of images processed using 16-bit AxAs. (a) PSNR. (b) MSSIM.

when K=8. However, when extended to 32 bits, its performance is inferior to that of AxPPLAs. The performance curves of AxPPLAs are generally positioned in the lower-left region relative to those of other AxAs, which demonstrates that AxPPLAs have better accuracy and energy efficiency. That is, AxPPLAs generally offer higher accuracy for the same energy expenditure. Specifically, compared to AxPPAs, the energy of AxPPLAs is reduced by up to 14.92% on average. It is reduced by up to 19.46% on average compared to all AxAs. When AxAs are extended to n=32 bits, the proposed AxPPLAs show greater advantages. Compared with all AxAs, the PDP is reduced by an average of up to 42.68%.

From the analyses above, MRED, MED, and NMED show similar trends for the considered AxAs, as can be seen in FIGURE 5. Thus, NMED will be compared for simplicity in what follows.

FIGURE 6 compares the values of PDP-NMED and EDP-NMED products for the AxAs. All these metrics are normalized using the corresponding values of LOAs. For the PDP-NMED product (in FIGURE 6(a)), AxPPLAs achieve a maximum reduction of up to 77.14%, compared to LOAs. For PPAs, these AxPPLAs exceed AxPPAs by up to 61.96% in terms of PDP-NMED product, especially pronounced at K=2. In contrast, ERCPAAs show the worst performance, with values far exceeding those of other AxAs. ETAs perform similarly to LOAs. APEx set-to-1 and AMA5 mirror operations reduce PDP effectively and slightly beat AxPPLAs at K=4 and 8. But when scaled to 32 bits proportionally, their advantage fades, and AxPPLAs perform better. Thus, the proposed AxPPLAs exhibit better tradeoffs compared to AxPPAs and other AxAs in PDP-NMED products at n=32 bits, regardless of the number of approximated bits (in FIGURE 6(c)). Similar trends are shown in EDP-NMED products for these AxAs (in FIGURE 6(b) and FIGURE 6(d)). For example, the most notable improvement is realized by the AxPPLAs when K=2 at n=16 bits, where the EDP-NMED product is reduced by 84.32% on average, compared to others. Specifically, under the two bit widths of n=16 bits and 32 bits, the proposed AxPPLAs reduce EDP-NMED by an average of 59.27% and 82.04%, respectively, compared with all mentioned AxAs.

5.4 | FoMs of AxAs

The main goal of AxAs is to lower hardware costs, which inevitably leads to some loss in accuracy. Therefore, to make a reasonable and fair comparison, overall performance metrics have to be taken into consideration, such as FoMs. This work utilizes $FoM1$ and $FoM2$ to evaluate AxAs as

$$FoM1 = \frac{PDP}{1 - NMED}, \tag{22}$$

$$FoM2 = Energy \times Delay \times Area \times NMED\,. \tag{23}$$

$FoM1$ uses the PDP and NMED of a design to measure accuracy and hardware performance, respectively[40]. It is suitable for evaluating the ability of a design to optimize power consumption and delay while meeting accuracy requirements. $FoM2$[17] comprehensively considers energy, delay, area, and accuracy (NMED), providing a holistic evaluation of multidimensional trade-offs.

These metrics are based on the goal of AxC by sacrificing some accuracy to optimize other hardware performances. They assess designs from the perspectives of accuracy-energy trade-offs and multidimensional efficiency to guide design methods to meet the requirements of various scenarios. According to these FoMs, AxAs with higher energy efficiency, faster speed, smaller area, and better accuracy will result in smaller values for $FoM1$ and $FoM2$.

FIGURE 7 compares the FoMs at different K to evaluate the overall performance of different AxAs in terms of hardware cost and accuracy. In the performance comparison of FIGURE 7(a) - FIGURE 7(b), AxPPLAs show significant advantages over most other adders (such as LEADxs, ERCPAAs, and LOAs), especially in scenarios with small values of K. Specifically, when the data bit width n=16 bits and K=2, the FoM1 and FoM2 of AxPPLAs are on average reduced by 31.32% and 80.04% compared to all the mentioned AxAs. Although AxPPLAs perform slightly worse than AMA5s and DSLLAs at K=8, when the data bit width is extended to n=32 bits, FIGURE 7(c) - FIGURE 7(d) show that AxPPLAs not only match the performance of other high-performance adders but even surpass them. These characteristics reveal that AxPPLAs exhibit strong adaptability

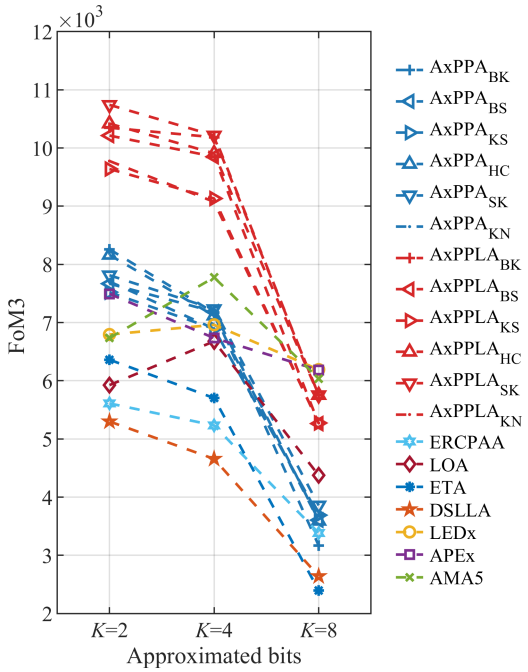

FIGURE 9 FoM3 of various 16-bit AxAs.

and development potential in different computing scenarios. For n=32 bits, the performance curves of AxPPLAs generally lie in a lower region than other AxAs. Their FoM1 and FoM2 values are reduced by 43.47% and 67.85% on average compared to all AxAs.

5.5 | Applications

The discussed AxAs with different approximated bits are applied in the Gaussian smoothing algorithm. Peak signal-to-noise ratio (PSNR) and mean structural similarity measure (MSSIM) are used to evaluate the image quality processed. *PSNR* is used to quantify the signal-to-noise ratio between exact and approximate results and is defined as[41]

$$PSNR = 10\log_{10}\left(\frac{h \times l \times MAX^2}{\sum_{i=0}^{h-1}\sum_{j=0}^{l-1}\left[I(i,j)-J(i,j)\right]^2}\right), \tag{24}$$

where MAX denotes the maximum pixel value of an image processed, h and l denote the dimensions of an image, and $I(i,j)$ and $J(i,j)$ denote exact and approximate results. *SSIM* evaluates the structural similarity between exact and approximate images processed as[42]

$$MSSIM(X,Y) = \frac{1}{N}\sum_{i=1}^{N}\frac{\left(2\mu_x\mu_y + C_1\right)\left(2\sigma_{xy} + C_2\right)}{\left(\mu_x^2 + \mu_y^2 + C_1\right)\left(\sigma_x^2 + \sigma_y^2 + C_2\right)}, \tag{25}$$

where N is the number of windows in an image, x and y refer to the window sizes for exact and approximate images, μ_x and μ_y, σ^2_x and σ^2_y, and σ_{xy} are the mean, variance, and covariance of exact and approximate images, respectively. C_1 and C_2 are constants added to the formula to stabilize the division, especially when the denominator is close to zero. Higher *PSNR* and *MSSIM* values suggest better image quality and higher similarity between exact and approximate images.

Five images from[43], including houseRGB, treeRGB, house, woman, and jelly beansRGB, are smoothened through a Gaussian smoothing kernel as

$$M = \frac{1}{60}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 4 & 4 & 4 & 1 \\ 1 & 4 & 12 & 4 & 1 \\ 1 & 4 & 4 & 4 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{26}$$

These images are used to obtain the average *PSNR* and *MSSIM* values of smoothened images using the considered AxAs with different approximated bits (K=2, 4, and 8 at n=16 bits) as shown in FIGURE 8. When the number of approximated bits is small, the PSNR values of all AxAs are close to 60dB, and the MSSIM values are close to 1. It means that the processed image quality is almost the same as the exact one. As the number of approximated bits increases, the image quality of the proposed AxPPLAs drops slower than others, only slightly lower than low-error architectures like LEDx. For example, when K=8, the *PSNR* value remains at 27.49dB, and the *MSSIM* value is close to 0.87 for images smoothened using the proposed AxAs. The average *PSNR* value of images processed by AxPPLAs is 46.76dB and the average *MSSIM* value is 0.96. Compared to other AxAs, the *PSNR* value has increased by 27.74% and the *MSSIM* value ranks third among these AxAs.

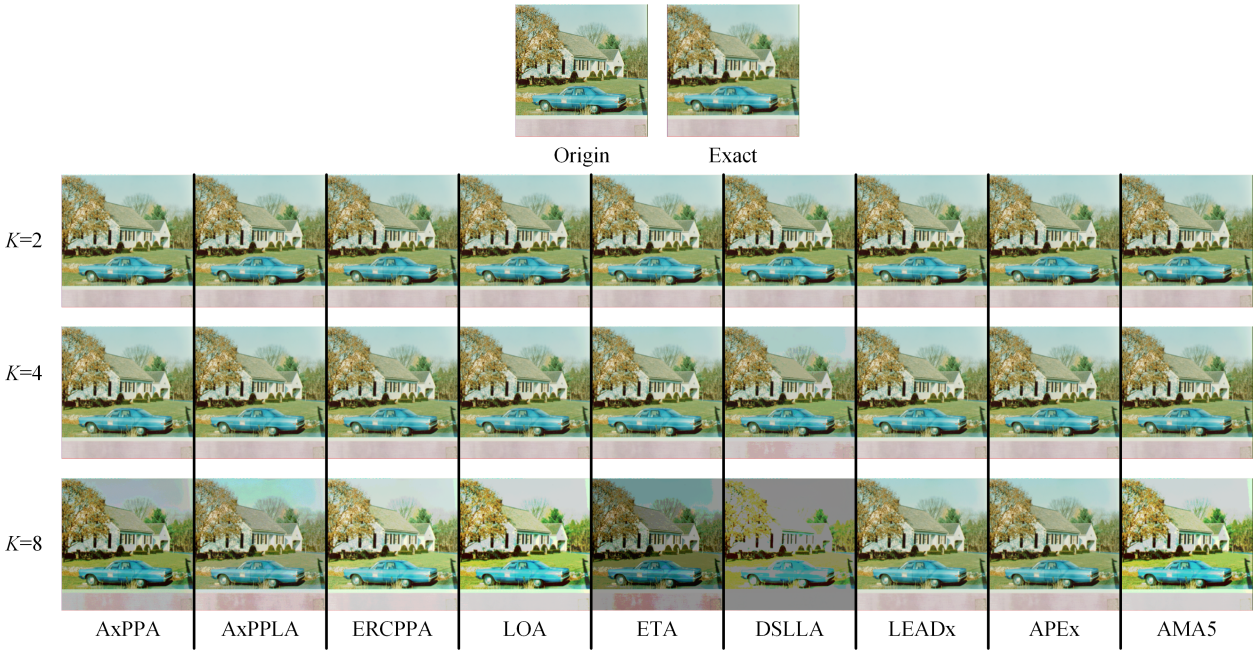

FIGURE 10 Examples of smoothened images(HouseRGB) using 16-bit AxAs.

In real applications, hardware cost also needs to be considered. To further enable a reasonable and fair comparison, this work employs *FoM*3 to evaluate the balance between application effects and hardware cost as[44]

$$FoM3 = \frac{PSNR \times MSSIM}{PDP}. \tag{27}$$

FIGURE 9 compares *FoM*3 at *K*=2, 4, and 8 to evaluate the overall performance of different AxAs in terms of hardware cost and application effects. The performance curves of AxPPLAs are usually in a higher region than other AxAs. When *K*=8, they are slightly lower than APExs and AMA5s, ranking third among all AxAs. But overall, AxPPLAs show a more advantageous performance level under different approximated bits (*K*=2, 4, and 8). Their *FoM*3 values are 45.44% upper on average compared to all other AxAs. They maintain a good balance between application effects and hardware overhead as illustrated by *FoM*3, and demonstrate stronger adaptability and competitiveness in practical scenarios.

FIGURE 10 shows examples of smoothened images using the AxAs with different approximated bits (*K*=2, 4, and 8). The differences between the exact image and images processed using AxAs at *K*=2 are minimal. Consequently, these images are difficult to distinguish visually. However, when *K*=8, the images processed using AxPPLAs retain more details than others, particularly performing better in edge and texture areas. In contrast, ERCPPAs and AMA5s exhibit inferior image quality at *K*=8. Furthermore, ETAs, DSLLAs, and AxPPAs produce darker images with noticeably reduced details, clearly visible to the naked eye. Although Gaussian smoothing is used as a representative application in this work, the error characteristics of the proposed AxPPLAs indicate that their applicability is not limited to a single use case. The approximation is applied only to the least-significant bit region, while the higher-order bits remain exact. As a result, the induced errors are mainly confined to the low-order bits and remain within a tolerable range.

This type of error behavior is generally compatible with accumulation-oriented computations and limited-precision datapaths. Typical examples include digital signal processing (DSP) pipelines involving summation or averaging operations, as well as sensor data aggregation and preprocessing[45]. In these scenarios, input signals often contain inherent noise, and small perturbations in low-order bits have a limited impact on overall computational quality. Although the exact impact depends on the target application, this analysis suggests that the proposed AxPPLAs are not restricted to a single use case and are structurally aligned with a broader class of approximate computing applications.

6 | CONCLUSION

This work proposes an approximate architecture for Ling adders based on parallel prefix topology (AxPPLAs). Less significant bits are computed through designed approximate prefix operators (AxPOs) to optimize the logic expressions of Ling carries, while more significant bits are exactly computed. Six typical parallel prefix topologies with various approximated bits are implemented with this design method to achieve a tradeoff between computing accuracy and hardware costs. Experimental results show that the proposed 16-bit AxPPLAs with the same number of approximated bits significantly reduce delay and energy consumption by up to 38.51% and 19.46% on average. The performance improvements become more remarkable when the bit width of AxAs scales to 32 bits. Overall analyses and comparisons indicate that the proposed AxPPLAs achieve a favorable trade-off between hardware efficiency and accuracy with other AxAs.

Although the experimental results of this paper are mainly targeted at 16-bit and 32-bit adders, the proposed AxPPLA architecture is inherently scalable to larger word lengths (e.g., 64-bit). The approximation is only applied to a predefined lower-bit region and implemented by AxPOs, while the higher significant-bit section remains an accurate parallel-prefix Ling adder. Therefore, extending the adder bit width does not require modifying the overall architecture, nor will it affect the

functional correctness of the accurate section. From a timing perspective, the shortened critical path achieved by simplifying prefix calculations and post-processing logic in the approximate region grows naturally as the bit width increases. Thus, the relative timing advantage brought by the proposed approximation method is expected to be sustained in adders with larger bit widths. In addition, the experimental results indicate that, under the same design methodology, the 32-bit adder implementations achieve overall performance trade-offs that are comparable to, and in some cases better than, those of the 16-bit designs. This trend further demonstrates that the proposed AxPPLA approach is not limited to small word sizes and is applicable to wider adders required in modern processors and accelerators.

ACKNOWLEDGMENT

This work was supported in part by the National Natural Science Foundation of China under Grant 62404067, in part by the Fundamental Research Funds for the Central Universities under Grant JZ2025HGTB0231 and PA2025GDSK0034, and in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada under Grant RES0048688.

DATA AVAILABILITY STATEMENT

Research data are not shared.

ORCID

Yongqiang Zhang https://orcid.org/0000-0003-1403-9128

REFERENCES

1. E. Napoli, E. Zacharelos, A. Strollo, and G. Meo, “Approximate full-adders: A comprehensive analysis,” *IEEE Access,* vol. 12, pp. 136054-136072, Sep. 2024.
2. H. Seo, and Y. Kim, “A low latency approximate adder design based on dual sub-adders with error recovery,” IEEE Trans. Emerging Top. Comput., vol. 11, no. 3, pp. 811-816, Jul.-Sep. 2023.
3. A. Dalloo, A. Humaidi, A. Mhdawi, and H. Al-Raweshidy, “Approximate computing: Concepts, architectures, challenges, applications, and future directions,” IEEE Access, vol. 12, pp. 146022-146088, Oct. 2024
4. N. Irtija, I. Anagnostopoulos, G. Zervakis, E. Tsiropoulou, H. Amrouch, and J. Henkel, “Energy efficient edge computing enabled by satisfaction games and approximate computing,” IEEE Trans. Green Commun. Netw., vol. 6, no. 1, pp. 281-294, Mar. 2022
5. U. Kumar, S. Bharadwaj, A. Pattaje, S. Nambi, and S. Ahmed, “CAAM: Compressor-based adaptive approximate multiplier for neural network applications,” IEEE Embedded Sys. Lett., vol. 15, no. 3, pp. 117-120, Sep. 2023.
6. L. Krishna, A. Sk, J. Rao, S. Veeramachaneni, and N. Sk, “Energy-efficient approximate multiplier design with lesser error rate using the probability-based approximate 4:2 compressor,” IEEE Embedded Sys. Lett., vol. 16, no. 2, pp. 134-137, Jun. 2024.
7. A. Kumar, S. Chatterjee, and S. Ahmed, “Low-power compressor-based approximate multipliers with error correcting module,” IEEE Embedded Sys. Lett., vol. 14, no. 2, pp. 59-62, Jun. 2022.
8. Y. Wu, H. Jiang, Z. Ma, P. Gou, Y. Lu, J. Han, S. Yin, S. Wei, and L. Liu, “An energy-efficient approximate divider based on logarithmic conversion and piecewise constant approximation,” IEEE Trans. Circuits Syst. I Regul. Pap., pp. 1-14, Apr. 2022.
9. C. Wu, W. Shi, Y. Yuan, Z. Zou, Z. Mo, and J. He, “Area-delay-energy-efficient approximate dividers based on piecewise linear fitting of surface,” IEEE Trans. Circuits Syst. I Regul. Pap., vol. 71, no. 11, pp. 5017-5029, Jul. 2024.
10. W. Liu, T. Xu, J. Li, C. Wang, P. Montuschi, and F. Lombardi, “Design of unsigned approximate hybrid dividers based on restoring array and logarithmic dividers,” IEEE Trans. Emerging Top. Comput., vol. 10, no. 1, pp. 339-350, Sep. 2022.
11. K. Chen, C. Xu, H. Waris, W. Liu, P. Montuschi, and F. Lombardi, “Exact and approximate squarers for error-tolerant applications,” IEEE Trans. Comput., vol. 72, no. 7, pp. 2120-2126, Dec. 2022.
12. M. Rosa, G. Paim, J. Godínez, E. Costa, R. Soares, and S. Bampi, “AxRSU: Approximate radix-4 squarer unit,” in IEEE International Symposium on Circuits and Systems (ISCAS), Austin, TX, USA, 2022, pp. 1655-1659.
13. L. Bandil, and B. Nagar, “Hardware implementation of unsigned approximate hybrid square rooters for error-resilient applications,” IEEE Trans. Comput., vol. 73, no. 12, pp. 2734-2746, Sep. 2024.
14. M. Rosa, G. Paim, P. Costa, E. Costa, R. Soares, and S. Bampi, “AxPPA: Approximate parallel prefix adders,” IEEE Trans. Very Large Scale Integr. VLSI Syst., vol. 31, no. 1, pp. 17-28, Jan. 2023.
15. H. Mahdiani, A. Ahmadi, S. Fakhraie, and C. Lucas, “Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications,” IEEE Trans. Circuits Syst. I Regul. Pap., vol. 57, no. 4, pp. 850-862, Apr. 2010.
16. N. Zhu, W. Goh, W. Zhang, K. Yeo, and Z. Kong, “Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing,” IEEE Trans. Very Large Scale Integr. VLSI Syst., vol. 18, no. 8, pp. 1225-1229, Oct. 2010.
17. J. Lee, H. Seo, H. Seok, and Y. Kim, “A novel approximate adder design using error reduced carry prediction and constant truncation,” IEEE Access, vol. 9, pp. 119939-119953, Sep. 2021.
18. W. Ahmad, B. Ayrancioglu, and I. Hamzaoglu, “Low error efficient approximate adders for FPGAs,” IEEE Access, vol. 9, pp. 117232-117243, Aug. 2021.
19. V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, “Low-power digital signal processing using approximate adders,” IEEE Trans. Comput. Aided Des. Integr. Circuits Syst., vol. 32, no. 1, pp. 124-137, Jan. 2013.
20. D. Esposito, D. De, E. Napoli, N. Petra, and A. Strollo, “Variable latency speculative Han-Carlson adder,” IEEE Trans. Circuits Syst. I Regul. Pap., vol. 62, no. 5, pp. 1353-1361, May. 2015.
21. D. Esposito, D. De, and A. Strollo, “Variable latency speculative parallel prefix adders for unsigned and signed operands,” IEEE Trans. Circuits Syst. I Regul. Pap., vol. 63, no. 8, pp. 1200-1209, Aug. 2016.
22. T. Karimi and A. Kamran, "Energy-delay efficient segmented approximate adder with smart chaining," IEEE Trans. Comput., vol. 74, no. 2, pp. 597-608, Feb. 2025
23. R. Doran, “Variants of an improved carry look-ahead adder,” IEEE Trans. Comput., vol. 37, no. 9, pp. 1110-1113, Sep. 1988.
24. R. Brent, and H. Kung, “A regular layout for parallel adders,” IEEE Trans. Comput., vol. C-31, no. 3, pp. 260-264, Mar. 1982.
25. P. Kogge, and H. Stone, “A parallel algorithm for the efficient solution of a general class of recurrence equations,” IEEE Trans. Comput., vol. C-22, no. 8, pp. 786-793, Aug. 1973.
26. A. Smith, and C. Lim, “Parallel prefix adder design,” in 15th IEEE Symposium on Computer Arithmetic, Vail, CO, USA, 2001, pp. 218-225.
27. T. Han, and D. Carlson, “Fast area-efficient VLSI adders,” in 1987 IEEE 8th Symposium on Computer Arithmetic (ARITH), Como, Italy, 1987, pp. 49-56.
28. S. Knowles, “A family of adders,” in Proceedings 14th IEEE Symposium on Computer Arithmetic (Cat. No.99CB36336), Adelaide, SA, Australia, 1999, pp. 30-34.
29. J. Sklansky, “Conditional-sum addition logic,” IRE Trans. Electron. Comput., vol. EC-9, no. 2, pp. 226-231, Jun. 1960.

30. A. Stefanidis, I. Zoumpoulidou, D. Filippas, G. Dimitrakopoulos, and G. Sirakoulis, “Synthesis of approximate parallel-prefix adders,” IEEE Trans. Very Large Scale Integr. VLSI Syst., vol. 31, no. 11, pp. 1686-1699, Nov. 2023.
31. Gulafshan, M. Khan, and M. Hasan, “Design of high speed, energy, and area efficient spin-based hybrid MTJ/CMOS and CMOS only approximate adders,” IEEE Trans. Magn., vol. 58, no. 5, pp. 1-8, May. 2022.
32. S. Kaza, S. Yarlagadda, and S. Maheswaran, “Ultra-low power approximate arithmetic circuits,” in 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), Delhi, India, 2023, pp. 1-5.
33. R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, “On reconfiguration-oriented approximate adder design and its application,” in IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Jose, CA, USA, 2013, pp. 48-54.
34. M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, “A low latency generic accuracy configurable adder,” in 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 2015, pp. 1-6.
35. M. Macedo, L. Soares, B. Silveira, C. Diniz, and E. Costa, “Exploring the use of parallel prefix adder topologies into approximate adder circuits,” in 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Batumi, Georgia, 2017, pp. 298-301.
36. P. Pereira, G. Paim, G. Ferreira, E. Costa, S. Almeida, and S. Bampi, “Exploring approximate adders for power-efficient harmonics elimination hardware architectures,” in IEEE 12th Latin America Symposium on Circuits and System (LASCAS), Arequipa, Peru, 2021, pp. 1-4.
37. A. Kahng, and S. Kang, “Accuracy-configurable adder for approximate arithmetic designs,” in DAC Design Automation Conference 2012, San Francisco, CA, USA, 2012, pp. 820-825.
38. K. Du, P. Varman, and K. Mohanram, “High performance reliable variable latency carry select addition,” in Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 2012, pp. 1257-1262.
39. J. Liang, J. Han, and F. Lombardi, “New metrics for the reliability of approximate and probabilistic adders,” IEEE Trans. Comput., vol. 62, no. 9, pp. 1760-1771, Sep. 2013.
40. F. Sabetzadeh, M. Moaiyeri, and M. Ahmadinejad, “A majority-based imprecise multiplier for ultra-efficient approximate image multiplication,” IEEE Trans. Circuits Syst. I-Regul. Pap., vol. 66, no. 11, pp. 4200-4208, Nov. 2019.
41. M. Martini, “A simple relationship between SSIM and PSNR for DCT-based compressed images and video: SSIM as content-aware PSNR,” in 2023 IEEE 25th International Workshop on Multimedia Signal Processing (MMSP), Poitiers, France, 2023, pp. 1-5.
42. Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” IEEE Trans. Image Process., vol. 13, no. 4, pp. 600-612, Apr. 2004.
43. "The USC-SIPI image database." 1981. [Online]. Available: https://sipi.usc.edu/database/.
44. D. Rostami, M. Eshghi, and Y. S. Mehrabani, “Low-power and high-speed approximate 4:2 compressors for image multiplication applications in cnfets,” Int. J. Electron., vol. 108, no. 8, pp. 1288-1308, Aug. 2021.
45. B. Mithil and C. Ramesh, "Approximate parallel prefix adders for image processing applications," in 2025 IEEE 5th International Conference on VLSI Systems, Architecture, Technology and Applications (VLSI SATA), Bangalore, India, 2025, pp. 1-6.