UNIVERSITY OF CALIFORNIA
RIVERSIDE

Approximate and Stochastic Circuit Design With Improved Accuracy and Efficiency

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering

by

Shuyuan Yu

March 2023

The Dissertation of Shuyuan Yu is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I wish to express my thankfulness to all the people who gave me the support and help through the whole five years of my PhD studies.

First and foremost, I want to express my deepest appreciation to my advisor Dr. Sheldon Tan for his guidance and help these years. His precious research experience, theoretical knowledge and kind mentorship is invaluable to my PhD research and future work.

I would also like to thank the committee members, Dr. Shaolei Ren and Dr. Daniel Wong for their suggestions and directions to my research when I started to explore the research fields at the very beginning.

Besides, I really feel grateful to all the other members in the VLSI Systems and Computation Lab: Hengyang Zhao, Chase Cook, Zeyu Sun, Han Zhou, Shaoyi Peng, Sheriff Sadiqbatcha, Jinwei Zhang, Wentian Jin, Yibo Liu, Liang Chen, Mohammadamir Kavousi, Maliha Tasnim, Chinmay Raje, Jincong Lu, Sachin Sachdeva, Subed Lamichhane for their collaboration and support in the projects during my PhD studies. Without the company of yours, it will be really hard time to go through these years in a foreign country.

The content of this thesis is reprinted or rewritten from the following published materials:

- Yu, Shuyuan, Yibo Liu, and Sheldon X-D. Tan. "COSAIM: Counter-based stochastic-behaving approximate integer multiplier for deep neural networks." 2021 58th ACM/IEEE Design Automation Conference (DAC). IEEE, 2021. (Chapter 2)

- Yu, Shuyuan, and Sheldon X-D. Tan. “Scaled-CBSC: scaled counting-based stochastic computing multiplication for improved accuracy.” Proceedings of the 59th ACM/IEEE Design Automation Conference. 2022. (Chapter 2)
- Yu, Shuyuan, Yibo Liu, and Sheldon X-D. Tan. “Approximate divider design based on counting-based stochastic computing division.” 2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD). IEEE, 2021. (Chapter 3)
- Yu, Shuyuan, Maliha Tasnim, and Sheldon X-D. Tan. “HEALM: Hardware-Efficient Approximate Logarithmic Multiplier with Reduced Error.” 2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, 2022. (Chapter 4)
- Yu, Shuyuan, and Sheldon X-D. Tan. “PAALM: Power Density Aware Approximate Logarithmic Multiplier Design.” Proceedings of the 28th Asia and South Pacific Design Automation Conference. 2023. (Chapter 5)

To my parents for all the support.

ABSTRACT OF THE DISSERTATION

Approximate and Stochastic Circuit Design With Improved Accuracy and Efficiency

by

Shuyuan Yu

Doctor of Philosophy, Graduate Program in Electrical Engineering
University of California, Riverside, March 2023
Dr. Sheldon Tan, Chairperson

Approximate computing enables efficient trade-off among accuracy, area, latency and power for more efficient error tolerant applications implementation such as machine learning and multimedia workloads. Those workloads are heavily dominated by the arithmetic operations and hence designs of hardware-efficient approximate arithmetic units have been intensively investigated recently. However, most of the existing works lack the systematic configurability for accuracy vs. area/power/latency trade-off. This thesis merges several works into designs of fast approximate arithmetic units with improved accuracy and hardware efficiency. The first work proposes a new counter-based stochastic-behaving approximate integer unsigned multiplier and scaling method with improved counting efficiency for many emerging error tolerant application workloads such as deep neural networks, mitigating a long-standing issue of the stochastic computing that small inputs usually lead to large error. The second work proposes a novel counting-based SC divider with accelerated counting process and improved accuracy to easily interface with the existing stochastic or binary logic. The third work focuses on improving the accuracy and the hardware efficiency of the conventional approximate logarithmic multiplier at the same time with a novel error compensation scheme. Last but not least, the forth work mitigates an important problem that approximated designs might lead to unwanted higher temperature and related reliability issues due to the increased power density and proposes a power density aware approximate logarithmic multiplier design which can reduce the power density of the original approximate logarithmic multiplier design with no accuracy loss.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Approximate Multiplication

Approximate computing enables efficient trade-off among accuracy, area, latency and power for more efficient error tolerant applications implementation such as machine learning and multimedia workloads [49, 37, 19, 18, 44, 16, 26, 27, 25, 7, 42, 50, 12, 13]. Those workloads are heavily dominated by the multiplication operations and hence design of hardware-efficient multiplier has been intensively investigated recently. The primary goal of the approximate multiplier design is to reduce the power and area for the least accuracy loss.

Multipliers are one of the most energy-hungry units. Area and the energy efficient multiplier design is paramount important for efficient processing of many machine learning and image processing workloads which require intensive multiply-accumulate operation (MAC) for matrix/tensor based operations [43]. In addition to the error-resilient nature, studies show that these workloads are robust to reduction in the precision of the arithmetic operations. 16-bit fixed point is demonstrated to be sufficient for training neural networks with no loss in classification accuracy [14]. 8-bit precision is sufficient for inference with minimal accuracy error loss[48]. Based on the aforementioned studies, a number of approximate multiplier designs have been proposed recently [24, 4, 36, 47, 46, 15, 34, 38, 39]. Those approximate multipliers employ some ad-hoc truncation or reduction methods or mathematically formulated approximation schemes. Most of the existing methods, however, lack the systematic configurability for accuracy vs. area/power/latency trade-off.

On the other hand, a class of approximate multipliers that are mathematically formulated include logarithmic multipliers, which convert multiplication into only shift and addition operations. Due to the inherent approximate nature of logarithmic operation and the easy accuracy manipulation of the resulting addition, the area, latency and power can be traded off at the cost of accuracy. The logarithmic multiplier was originally proposed by Michelle [31]. Since then, many approximate logarithmic multipliers (ALM) have been proposed to improve Michelle's work [38, 30, 3, 39]. Most of those methods focused on how to reduce and compensate the errors introduced in the piece-wise approximation of the log function, which tends to cause negative errors.

Recently Ansari *et al.* [3] developed an approximate scheme to make the error distribution more balanced (double sided errors) for the ALM method. Saadat *et al.* [39] further introduced a general error compensation technique, called *REALM*, using an analytically generated error reduction factor lookup table for different regions of input operands. The benefits of this method are that it can generate more balanced errors by design and provide configurable design trade-off between area and precision. However, this method use one lookup table for all the truncation configuration in the approximate addition, which may lead to large errors especially for low precision cases.

1.2 Stochastic Computing (SC)

Stochastic computing (SC), which is an important branch of approximate computing, is a promising low-cost, error-resilient alternative to the conventional binary based computing. Compared to binary design, SC is shown to have better error resilience, progressive trade-off among performance, accuracy and energy, as well as cheap implementation of complex arithmetic operations.

1.2.1 SC Multiplication

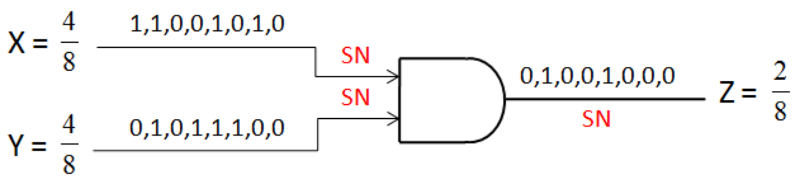


Figure 1.1: Conventional SC multiplication.

Fig. 1.1 shows the conventional SC multiplier, where the number or stochastic number (SN), is represented by a bit-stream, whose signal probability, or frequency of bit '1', determines its value. Naturally, the value is defined in the range $[0, 1]$, called unipolar, or over $[-1, 1]$, called bipolar. For instance, in Fig. 1.1, the number X represents $4/8$ as we have four bit '1's in the bit-stream of 8 bits. One of the major benefits for SC is that many arithmetic operations such as multiplication can be simply implemented by AND operation (or *XNOR* gate for bipolar) as shown in this figure. SC multiplication has been applied to error-correcting codes [32], image processing [26], and recently deep neural networks (DNNs) [20, 41, 40, 17].

However, SC multiplication's simple hardware implementation suffers from several drawbacks: first, traditional SC takes 2^N cycles to deal with N-bit precision inputs (as shown in Fig. 1.1), which can be much larger than binary logic especially when N is large. So SC multiplication is still more suitable for applications which don't require high accuracy. Second, the accuracy of SC multiplication process requires the randomness of two bit-streams (ideally with zero correlation) in addition to the length of bit-stream. As a result, many research works have been proposed to develop high-quality random number generators (RNGs) that exhibit zero or close to zero correlation, including low-discrepancy sequences [28, 29], bit scrambling methods [35, 21].

1.2.2 SC Division

In traditional binary arithmetic, division is somewhat more complex than multiplication, but broadly similar circuits can be used for both operations, e.g., combinational arrays or sequential shift-and-add/subtract circuits [22]. This is not the case for stochastic division, however. No practical combinational divider for SC is known, and sequential division relies on iterative add/subtract structures that loosely resemble binary dividers, e.g., ADDIE-based dividers [10], which are examined later. Recalling the divide-by-zero problem of binary division, the restricted [0, 1] range of SC is also problematic because the quotient P_{X_1}/P_{X_2} approaches $\pm\infty$ as the divisor P_{X_2} goes to zero. Depending on the implementation, the results of both binary and SC division can lie between zero and the largest representable number.

During the past several decades, many SC division approaches have been proposed. These approaches can be briefly divided into three kinds: designs which require independent input bit-streams, designs which require highly correlated input bit-streams, and designs which have no requirement on the bit-stream dependence or correlation. In early time, the accuracy of almost all SC circuits depends on having input SNs that are uncorrelated, i.e., statistically independent. Recent studies showed that the SC division could be performed much more efficiently when the two input bit-streams were highly correlated so the division could be approximated by computing the conditional probability of the two streams [5, 51]. To mitigate the different correlation requirements between traditional SC division, a translation module (called skewed synchronizer) was introduced to resolve this problem at the cost of higher area overhead [51].

1.3 Related Works

1.3.1 SC Multiplication

SC multiplication usually consists of 3 parts: (1) stochastic number generators (SNG) which convert the N-bit binary inputs to the 2^N-bit stochastic numbers (SN); (2) SC multiplication core, usually an *AND* or *NOR* gate which is corresponding to unsigned multiplication or signed multiplication; (3) a counter which converts the product SN to binary form back again if needed.

As the computing accuracy of SC multiplication is determined by the SN quality, in other words, the SNG. And the SNG also costs much more area and power than the SC multiplication core. Existing works mainly focused on how to design more area efficient and high quality SNGs, like: Halton sequence generator [2], LFSR (linear feedback shift register) [1], LD (low discrepancy) sequence generation [33].

Among these works, a recent work proposed by Sim [40], named counting-based SC (CBSC) multiplication achieved not only better accuracy but also a smaller latency by introducing an finite state machine (FSM) based SNG with counting scheme. To further improve the computing latency of CBSC method, work in [6] exploits the symmetric properties of the deterministic bit stream pattern so that one can start the counting process from either the end or the beginning of the SN bit-stream depending on the value of the weight to reduce the clock cycles needed.

These aforementioned works indeed helped reduce the latency, but the long-standing low accuracy issue for small numbers for SC still remains. Basically this is due to the intrinsic SC multiplication property that two N-bit inputs for SC multiplication still generate N-bit output product instead of $2N$-bit output.

To mitigate this issue, Zhou in [53] has proposed a scaled population (SP) arithmetic concept. This method indeed improved the error metrics in the "small" range by inserting bit '1's into the input SN bit-stream. But SP method still suffers from a few problems like: difficult to interface with the binary logic; requiring complicated design to reduce random fluctuation errors; hard to represent the scaling term; and contradiction between the low bit ‘1’ density favor of SC addition and the high bit ‘1’ requirement of SC multiplication.

1.3.2 SC Division

Traditional SC division follows Gaines’ design [10], in which an up-down counter is used to reach an equilibrium when two SN numbers, $x_{1,SN}$ and $x_{2,SN} \cdot p_{z,SN}$ are equal. Then $p_{z,SN} = x_{1,SN}/x_{2,SN}$. So it needs SC multiplication for $x_{2,SN} \cdot p_{z,SN}$. Gaines’ design was proved to take long time to converge [5] and required both the dividend and the divisor to be uncorrelated. A recent work proposed by Temenos [45] mitigated these issues by using a deterministic FSM-based stochastic division design (DFSM-DIV), which did not require the independence between the input SN bit-streams. However, DFSM-DIV still required at least 3 orders of magnitude (10^2) clock cycles to converge in average and could not achieve much accuracy improvement.

Another idea, which is correlation based, made improvement in a different way. *Correlated division*, or CORDIV, proposed by Te-Hsuan [5] explored the fact that division could be computed as the conditional probability of two SN numbers: $P_{X_1|X_2}$. This work proved that with highly correlated input SN bit-streams, CORDIV could obtain much more accurate results when compared with Gaines’ design [10]. But the accuracy of CORDIV method depends on the quality of input SN bit-streams.

Specially designed SNGs shown in Fig. 3.1 [5] has been proved that could mitigate this problem. However, the accuracy still depends on the distribution of bit ‘1’ in input SN bit-streams. Adding skewed synchronizer could improve accuracy, but was still not accurate enough, especially in certain output value ranges [51].

Besides, all these SC-based division designs suffered from accuracy varying among different output value ranges and the problem of large latency, taking 2^N cycles to finish the process with inputs whose corresponding binary precision are N-bit, which is an intrinsic shortcoming of SC implementation. Furthermore, the energy consumption is also high when the inputs and outputs are in binary form to interface with other computing parts in typical image process application [8].

1.3.3 Approximate Logarithmic Multiplication

Earlier approximate unsigned integer multiplier designs often involved ad-hoc based approximations, such as recursive multipliers [24] which consist of 2×2 multiplication blocks, simplification of Wallace tree [4], simplifying partial product generation/summation [36, 47, 46], others used a smaller multiplier by extracting m-bit fragment from the N-bit precision inputs. Such as [15, 34].

Among these methods, many recent approximate multipliers are developed based on the classic approximate logarithmic multiplier proposed by Mitchell, also called *ALM*, as it shows good overall performance and has flexibility for trade-offs among area, power and accuracy [31].

To further improve the accuracy of the ALM method, several derivative works have been proposed by means of different error compensation mechanisms. For instance, the MBM design tried to add a fixed single error-correction term to the final result [38]. This was further improved by the LeAp multiplier, which added different error coefficients to the fraction parts based on the value ranges of the results [9]. The REALM multiplier design further improved the compensation scheme by using a lookup table to store $M \times M$

coefficients / factors for $M \times M$ partitions of input ranges with some hardware resource overheads [39]. These works indeed improved the error metrics of the approximate logarithmic multiplication without incurring too much resource overheads.

One important observation is that the ALM design will become less effective in reducing area and power when the precision of inputs decreases. Ebrahimi *et al.* [9] recently showed that 32-bit ALM can have more area and power reduction than 16-bit ALM. However, low precision operation is important as emerging machine learning workloads can be performed (at least for inference) using low precision operations. For instance, 16-bit fixed point is demonstrated to be sufficient for training neural networks with no loss in classification accuracy [14]. 8-bit precision is sufficient for inference with minimal accuracy loss[48].

Some previous works [3, 30] tried to do further area reduction by replacing the exact adder with an inexact one. Since the exact adder unit is the bottleneck of the ALM critical path and occupies large area, this idea does help in area saving. But the inexact adder also introduces extra error, and the error can become quite significant especially in the 8-bit case.

1.4 Contributions

The work presented in this thesis presents several contributions in approximate arithmetic unit designs:

- A counter-based stochastic-behaving approximate integer multiplier (COSAIM) is proposed based on a state of art counting-based SC multiplication. Instead of counting the bit '1' sequentially, we propose to compute the number of '1' in the bit stream using a simple formula for different positions in corresponding to the binary number, which can significantly speed up the counting process and further lead to significant clock cycle reductions with no additional accuracy loss.

- The state of art counting-based SC multiplication and the proposed COSAIM design are further extended by introducing a scaling factor to the original frameworks. The bit '1' density in the bit-stream then is promised to be always large than 0.5 to ensure sufficient accuracy for SC computing and the scaling factor is guaranteed to be integer. Further more, the new scaled SC format is in the binary form and all the operations like shifting is done in the binary format, which is much more efficient than the bit-stream level operations.
- A new counting-based SC division design is developed based on a recent proposed correlation division design and deterministic bit-stream generation. The new design can be performed in a counting process, with reduced operation time and improved accuracy.
- A novel hardware efficient approximate logarithmic multiplier, named *HEALM*, with a novel error reduction scheme for low precision (8-bit to 16-bit) multiplication is proposed. With different error coefficients lookup tables for each error compensation or reduction case, the HEALM design shows better performance than previous works: more accurate result with both reduced area and power.
- The problem that recently proposed approximate logarithmic multipliers (ALMs) can actually lead to higher power density than the binary logic design with potential ther-

mal issues is demonstrated for the first time. And a power density aware approximate logarithmic multiplier (PAALM) design is proposed. The PAALM design profiles the switching activities of the existing ALM designs and try to reduce the high computing switch activities based on equivalent mathematical formula at some costs of area so that the power density can be reduced with no accuracy loss. PAALM can be viewed as a way to trade-off some area overhead for power density reduction, which is however necessary for reducing local hot spots and maintain the long-term reliability of chips.

1.5 Organization of This Thesis

The rest of this thesis is organized as follows: Chapter 2 introduces the proposed counter-based stochastic-behaving approximate integer multiplier (COSAIM) design and the scaled counting-based stochastic computing multiplication; Chapter 3 shows the details of the proposed counting-based stochastic computing division; Chapter 4 demonstrates a novel hardware efficient approximate logarithmic multiplier; and Chapter 5 shows a power density aware approximate logarithmic multiplication design. Finally, Chapter 6 summarizes the thesis.

Chapter 2

Counter-Based Stochastic-Behaving Approximate Integer Multiplier Design and Scaling Technique

2.1 Review of State of Art Works

a recent work proposed by Sim [40], named counting-based SC (CBSC) multiplication achieved not only the better accuracy but also the smallest latency by introducing a finite state machine (FSM) based SNG with counting scheme. The CBSC multiplication design is shown in Fig. 2.1. Fig. 2.1(a) demonstrates the FSM based SNG which generates SN bit-stream in a deterministic way. Fig. 2.1(b) shows the concept of the CBSC multi-

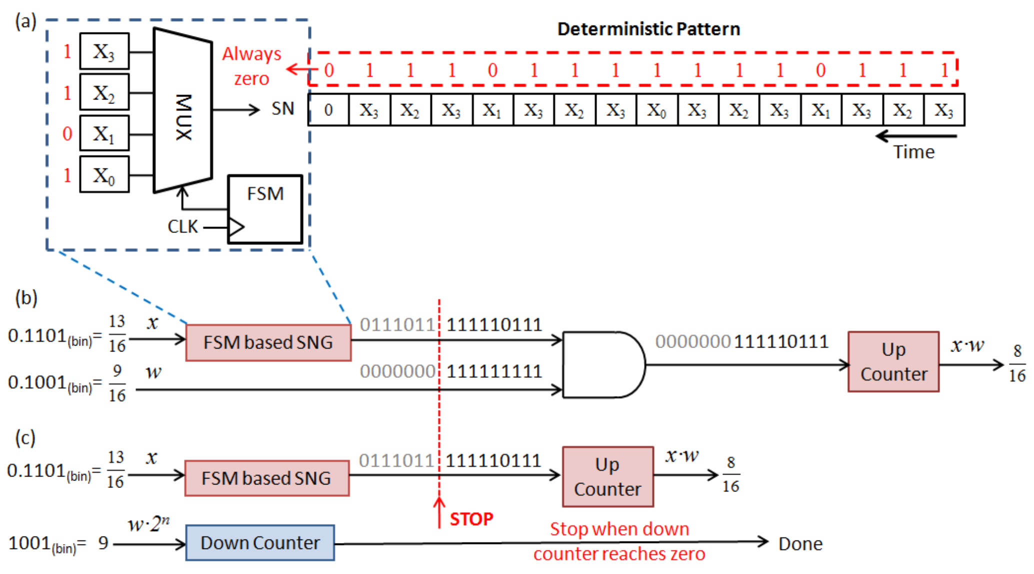


Figure 2.1: The CBSC multiplication design.

plication. While Fig. 2.1(c) illustrates the CBSC multiplication method. Different from the traditional SC multiplication, CBSC only requires one FSM-based SNG to convert one of the two binary inputs, ex. x, into a bit-stream with deterministic pattern first. The FSM-based SNG evenly distributes the x_{i-1}, which is the ith bit of x, based on its binary weight 2^{i-1}. For instance, if $i = 3$, then x_2 will appear 4 times in the resulting SN. Such SN generation can be simplified and implemented by an FSM and a MUX. The FSM is actually an up counter counts from 0 to $2^N - 1$, assuming x is N-bit. The MUX then outputs x_{i-1} based on the output value of the FSM.

If the SN bit-stream for the other input w is set to be series of '1' followed by a bunch of '0' as shown in Fig. 2.1. As SC multiplication is simply *AND* operation, it is no necessary to count the second half of the output bit-stream. So, the whole counting process only requires $w \cdot 2^N$ cycles to finish, which saves half latency in average. The authors used a down counter to realize the idea. While $w \cdot 2^N$ is used as the initial value, the down counter decreases by one in each clock cycle. When it reaches "zero", the process is terminated. As a result, CBSC leads to a simpler design as one traditional SNG (typically using LFSR) and the AND gate are removed in exchange of a down counter, which is much cheaper than an SNG.

To further improve the computing latency of CBSC method, work in [6] exploits the symmetric properties of the deterministic bit stream pattern so that one can start the counting process from either the end or the beginning of the SN bit-stream depending on the value of the weight to reduce the clock cycles needed. By exchanging the up counter which is used to convert the product SN to binary form to an up/down counter. And the initial value $Init$ of the down counter is determined by the most significant bit (MSB) as (2.1).

$$Init = \begin{cases} w \cdot 2^N, & w < 0.5, \\ 2^N - 1 - w \cdot 2^N, & w \geq 0.5 \end{cases} \tag{2.1}$$

The average computing latency is cut to 1/4 of the traditional SC multiplication.

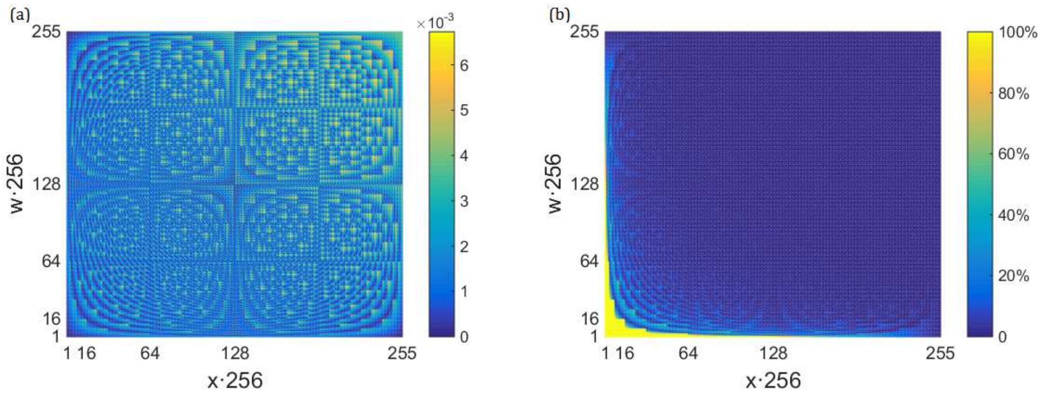


Figure 2.2: (a)The absolute error distribution of the counting-based SC method; (b)The relative error distribution of the counting-based SC method.

These aforementioned works indeed helped reduce the latency, but the long-standing low accuracy issue for small numbers as shown in Fig. 2.2 for SC still remains. Basically this is due to the intrinsic SC multiplication property that two N-bit inputs for SC multiplication still generate N-bit output product instead of $2N$-bit output.

To mitigate this issue, Zhou in [53] has proposed a scaled population (SP) arithmetic concept. This method indeed improved the error metrics in the "small" range by inserting bit '1's into the input SN bit-stream. As a result, the bit '1' density of the SN bit-stream is then raised to a certain level, like 0.7, for example. Since the SN value is changed during the bit '1' insertion process, the SP method introduced an exponent term so that the scaled SN number will remain approximately the same. The resulting SP number is a 2-tuple including a scaling term and a SN term. But SP method still suffers from a few problems. First, the method is still based on the traditional random-number based SC scheme. The format is difficult to interface with the commonly used binary number format, which is often required in the image processing and machine learning application. Second, it operates in the bit-stream level and requires complicated design to reduce random fluctuation errors for random number generation, scaling and random-shuffling operations. Third, it requires the bit '1' density to be any given value such as 0.7, which can lead to no integer number in the scaling term. Fourth, it uses the *OR* operation for SC addition, which however favors the low bit '1' density (small value) in the SC bit stream, and it contradicts the high bit '1' density requirement in the SC multiplication.

In this chapter, based on these state of art works, a counter-based stochastic-behaving approximate integer multiplier, or briefly *COSAIM*, and a scaled counting-based

SC multiplication approach, *Scaled-CBSC* are proposed to reduce the computing latency at the maximum level with minimum area overhead and mitigate the long-standing low accuracy issues of SC.

2.2 Proposed Counter-Based Approximate Multiplier

In this section, we elaborate our proposed integer approximate multiplier: *CO-SAIM*.

2.2.1 The Proposed Accelerated Counter-Based Multiplication

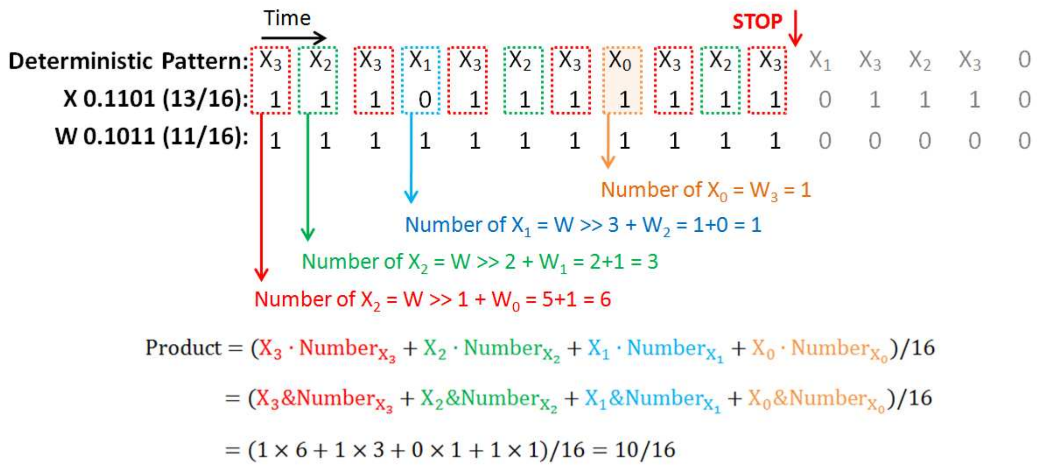
Suppose we have a stochastic number (SN) converted from a binary number x, x is N-bit. The product of CBSC-MUL equals to how many times the bit '1' appears in the stochastic number bit-stream of x in the first w cycles, while w is the other input. So the product of CBSC can also be represented as (2.2).

$$Product = \sum_{i=0}^{N-1} N_{x_i} \cdot x_i \tag{2.2}$$

Following the SN deterministic pattern of CBSC, x_{N-1}, which is the most significant bit (MSB), appears once every 2 cycles. Since the counting process stops at wth cycle, then we can easily prove that $N_{x_{N-1}}$ equals to $w/2$. When w is even, there is no remainder after $w/2$, we simply do 1-bit right shift operation to w to obtain $w/2$. But when w is odd, we need to count one more cycle, then $N_{x_{N-1}}$ equals to $\lfloor w/2 \rfloor + 1$. As w being odd or even is determined by w_0, then $N_{x_{N-1}}$ can be simply represented in the form of $\lfloor w/2 \rfloor + w_0$. In the same manner, as x_{N-2} appears once every 4 cycles, $N_{x_{N-2}}$ equals to $\lfloor w/4 \rfloor + w_1$, and finally arbitrary N_{x_i} can be calculated by (2.3).

$$N_{x_i} = \lfloor w/2^{N-i} \rfloor + w_{N-1-i} \tag{2.3}$$

To further illustrate this, we walk through an example of the 4-bit multiplication shown in Fig. 2.3.



Time
STOP
Deterministic Pattern: X3 X2 X3 X1 X3 X2 X3 X0 X3 X2 X3 X1 X3 X2 X3 0
X 0.1101 (13/16): 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0
W 0.1011 (11/16): 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
Number of X0 = W3 = 1
Number of X1 = W >> 3 + W2 = 1+0 = 1
Number of X2 = W >> 2 + W1 = 2+1 = 3
Number of X2 = W >> 1 + W0 = 5+1 = 6
Product = (X3 · NumberX3 + X2 · NumberX2 + X1 · NumberX1 + X0 · NumberX0)/16
= (X3&NumberX3 + X2&NumberX2 + X1&NumberX1 + X0&NumberX0)/16
= (1 × 6 + 1 × 3 + 0 × 1 + 1 × 1)/16 = 10/16

Figure 2.3: 4-bit approximate multiplication example.

2.2.2 The Circuit Design for the Proposed COSAIM Multiplier

We show the structure of our proposed COSAIM design in Fig. 2.4. The proposed design is composed of shift registers, *AND* Gates and adders. $\lfloor w/2^{N-i} \rfloor$ is realized by simply using bit shifting operation. After added by w_{N-1-i}, the number of x_i is obtained. As one bit multiplication is actually a simple *AND* operation, we use an *AND* Gate to obtain the value of $N_{x_i} \cdot x_i$. Then to sum up $N_{x_i} \cdot x_i$ together, we utilize an adder tree. The number of the adder tree levels is actually also the number of the pipeline stages. Increasing the number of the pipeline stages will increase the speed of the COSAIM design, but also increase the required area at the same time. For N-bit inputs, if we use the largest possible pipeline stages, the computation latency of the adder tree is of $\mathcal{O}(Log_2(N))$. That means we can do latency reconfigurable design based on the input bit-width just like stochastic computing. To achieve the highest computing speed, the number of of the adder tree stages is compressed to one in Sec. 2.4.

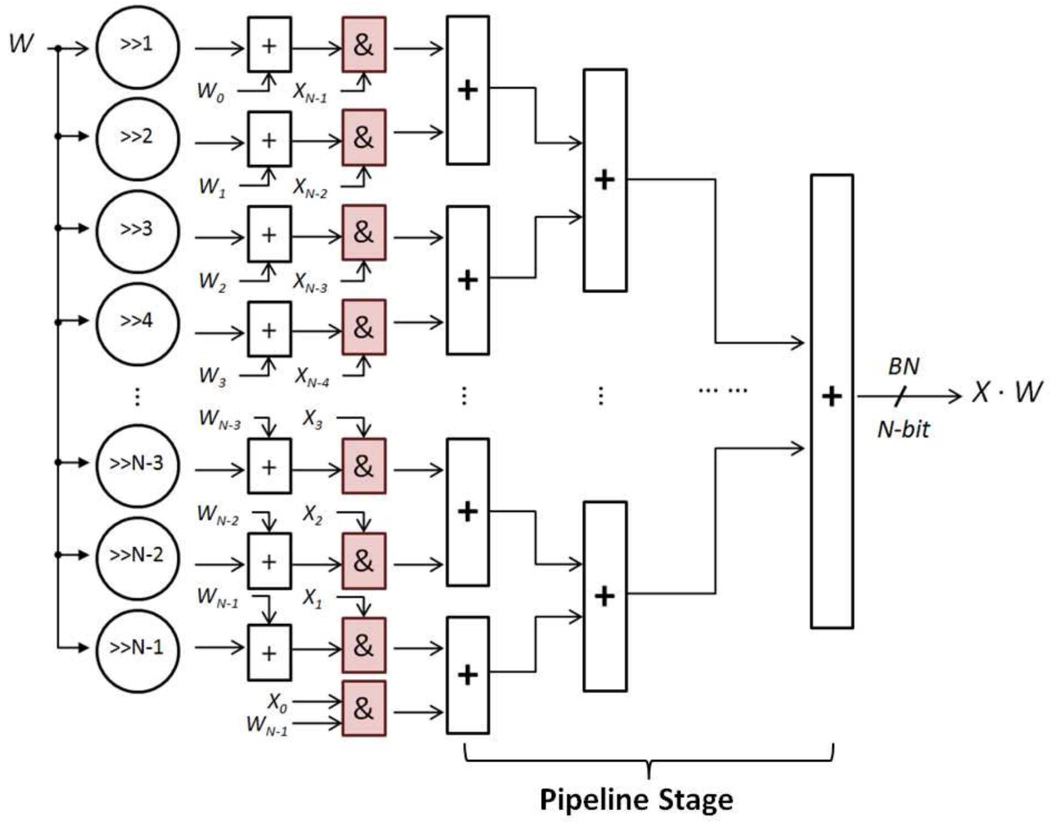


Figure 2.4: COSAIM design.

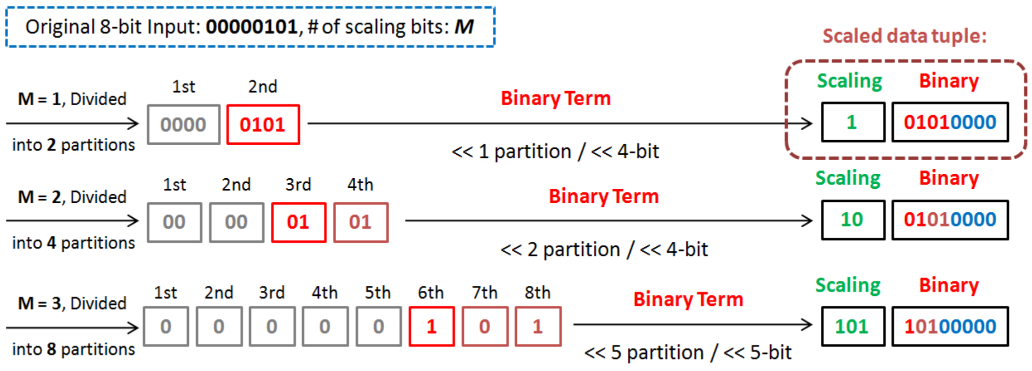
2.3 Proposed Scaled SC Multiplication Approach

In this section, we present the details of our proposed scaled counting-based SC (Scaled-CBSC) multiplication approach to mitigate the large errors for small number multiplication.

2.3.1 Scaled Counting-Based SC Multiplication Method

As shown in Fig. 2.2(b), SC multiplication will have very large relative error with “small” inputs, especially in the range of [1/256, 15/256], due to the low bit ‘1’ density of the input SN bit-stream, which is an intrinsic property of SC multiplication. So a simple idea to improve the error metrics of SC multiplication is to avoid these “small” inputs, in other words, increasing the bit ‘1’ density of the input SN bit-stream. Unlike SP method [53] which inserts bit ‘1’ into the SN bit-stream, our proposed Scaled-CBSC multiplication directly enlarge the binary number inputs, which is also equivalent to raising the bit ‘1’ density level of the SN bit-stream.

In the Scaled-CBSC multiplication, an N-bit binary number data is represented by a ‘scaling-binary’ 2-tuple. The tuple consists of two parts, an M-bit scaling term and an N-bit binary term. The data representation and how an N-bit binary number be converted to 2-tuple $\{M, N\}$ is shown in Fig. 2.5. Specifically the scaling term has M bits, the original binary number is then divided into 2^M partitions. To mitigate the SC multiplication error, we want the binary term to be as large as possible to avoid the “small” inputs regions. So the leading bit ‘1’ in the binary term, which is marked with red in Fig. 2.5, should appear in the most significant partition. We then left shift $N/2^M$ bits, or one partition each time. The



Original 8-bit Input: 00000101, # of scaling bits: M
Scaled data tuple:
M = 1, Divided into 2 partitions
1st
2nd
0000
0101
Binary Term
<< 1 partition / << 4-bit
Scaling
Binary
1
01010000
M = 2, Divided into 4 partitions
1st
2nd
3rd
4th
00
00
01
01
Binary Term
<< 2 partition / << 4-bit
Scaling
Binary
10
01010000
M = 3, Divided into 8 partitions
1st
2nd
3rd
4th
5th
6th
7th
8th
0
0
0
0
0
1
0
1
Binary Term
<< 5 partition / << 5-bit
Scaling
Binary
101
10100000

Figure 2.5: The data tuple including the scaling and the binary terms.

value of the scaling term records number of times the shift operation being carried out. We can easily derive that the maximum number of the scaling bits, or M_{MAX}, equals to $\log_2^N$. As the binary term will always promise the leading bit '1' in the most significant partition. Each partition consists of $N/2^M$ bits, and the least significant bit in the most significant partition weights $(1/2)^{N/2^M}$. Thus it can be verified that after bit partition-based shifting operations, the value of the binary term should be no less than $(1/2)^{N/2^M}$. We show an 8-bit example in Fig. 2.6. Since we use the binary term of the scaled data presentation as the input of the CBSC kernel in our proposed Scaled-CBSC method (will be discussed later in Sec. 2.3.2), our binary only representation is more compact than the SP format [53], which is a mixed 2-tuple of bit-stream and binary numbers.

The resulting Scaled-CBSC multiplication approach is shown in Fig. 2.7. The Scaled-CBSC multiplication consists of two blocks. One is the simple M-bit binary summation to sum the scaling terms (green blocks in Fig. 2.7) of the two input tuples (x and w) up. The other block is the CBSC kernel which is shown in Fig. 2.1. The inputs of the

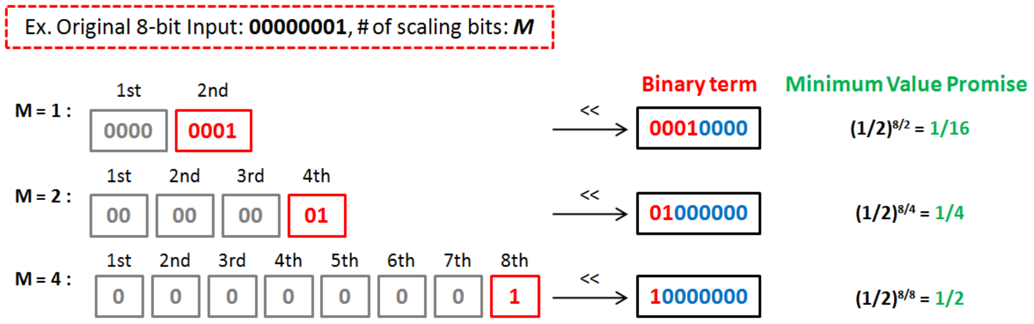


Figure 2.6: An 8-bit example to show the minimum promised value of the binary term.

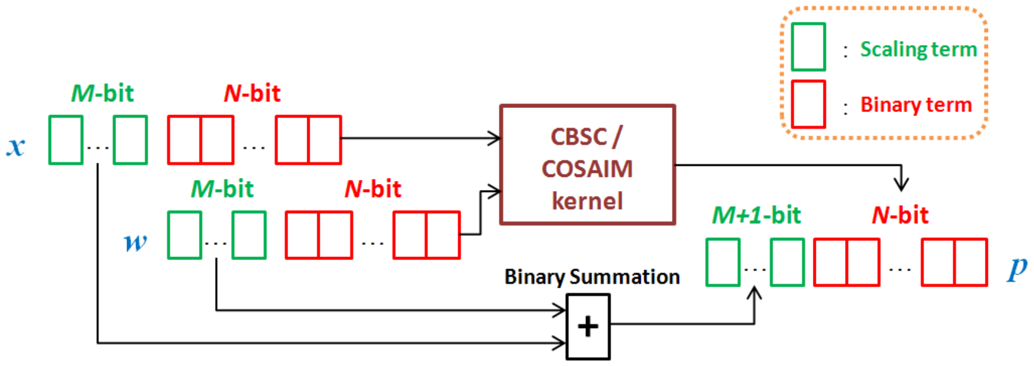


Figure 2.7: The Scaled-CBSC/COSAIM method.

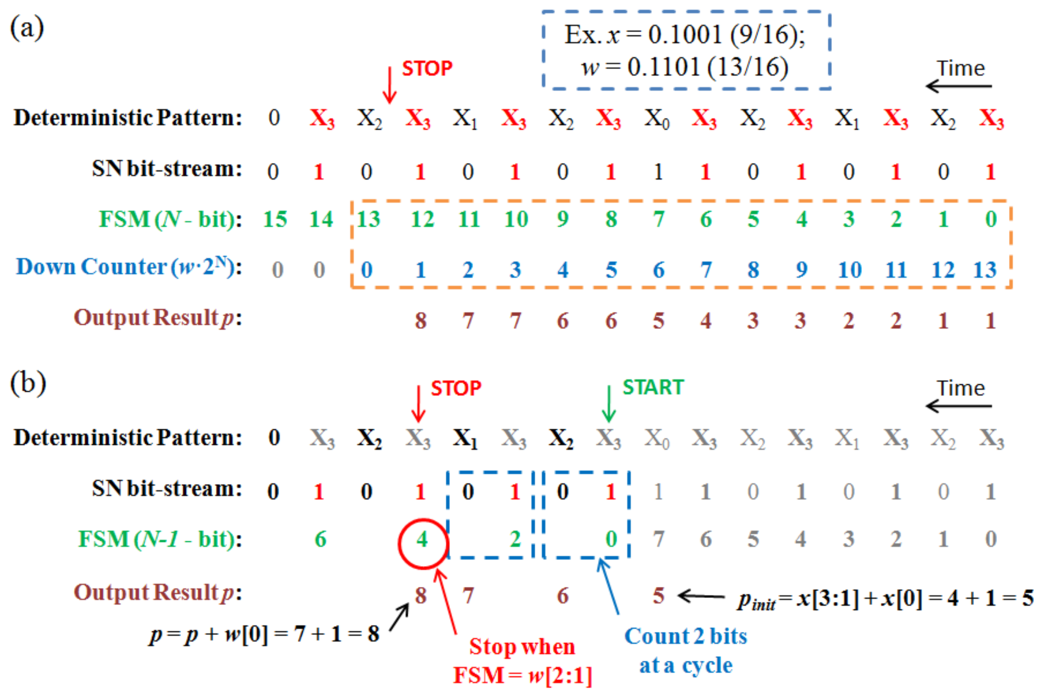

(a)
Ex. x = 0.1001 (9/16); w = 0.1101 (13/16)
STOP
Time
Deterministic Pattern: 0 X3 X2 X3 X1 X3 X2 X3 X0 X3 X2 X3 X1 X3 X2 X3
SN bit-stream: 0 1 0 1 0 1 0 1 1 1 0 1 0 1 0 1
FSM (N - bit): 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Down Counter (w·2^N): 0 0 0 1 2 3 4 5 6 7 8 9 10 11 12 13
Output Result p: 8 7 7 6 6 5 4 3 3 2 2 1 1
(b)
STOP
START
Time
Deterministic Pattern: 0 X3 X2 X3 X1 X3 X2 X3 X0 X3 X2 X3 X1 X3 X2 X3
SN bit-stream: 0 1 0 1 0 1 0 1 1 1 0 1 0 1 0 1
FSM (N-1 - bit): 6 4 2 0 7 6 5 4 3 2 1 0
Output Result p: 8 7 6 5
p = p + w[0] = 7 + 1 = 8
Stop when FSM = w[2:1]
Count 2 bits at a cycle
p_init = x[3:1] + x[0] = 4 + 1 = 5

Figure 2.8: (a) The output value of the original CBSC kernel components. (b) The output value of the optimized CBSC kernel components with the maximum number of the scaling bits.

CBSC kernel are the binary terms (red blocks in Fig. 2.7) of the two input data tuples. The output product p is also a 2-tuple, but with a $M+1$-bit scaling term and an N-bit binary term. Furthermore, the Scaled-CBSC multiplication could be further extended to *Scaled-COSAIM* with the CBSC kernel in Fig. 2.7 simply exchanged to COSAIM kernel proposed in Sec. 2.2.

2.3.2 Hardware Design and Optimization

Assume a $\{M, N\}$-bit 2-tuple has the maximum number of the scaling bits M_{MAX}, as mentioned in Sec. 2.3.1, the minimum value of the binary term should be 0.5. From Fig. 2.8(a), we notice that if w is promised to be no less than 0.5, the computing latency

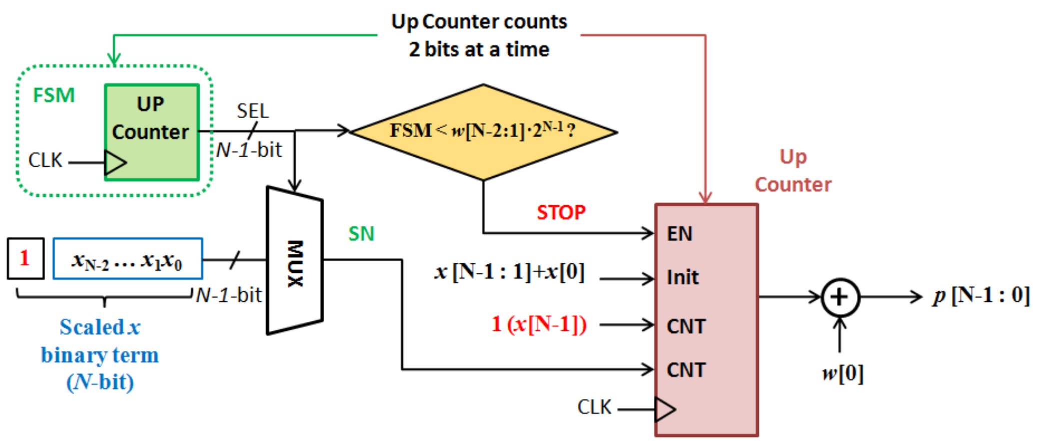


Figure 2.9: The structure of the optimized CBSC kernel with the maximum number of scaling bits.

of the SC multiplication should be no less than $2^N/2$. And since the SN bit-stream of x is "centrosymmetric", the up counter output value at the $2^N/2$th clock cycle always equals to $x/2 + x[0]$. So the counting process before the $2^N/2 + 1$th cycle is promised, which means we don't need to count the first $2^N/2$ cycles anymore. We can simply start the counting process from the $2^N/2 + 1$th cycle with an initial value of $x/2 + x[0]$ $(x[N-1:1] + x[0])$. Thus we can save $2^N/2$ clock cycles of the computing latency of CBSC method, which is shown in Fig. 2.8(b).

With the maximum number of scaling bits, the minimum value of the binary term of x equals to 0.5, which means the most significant bit (MSB) of x is bit '1'. Based on the deterministic SN generation pattern of CBSC method, bit '1' will appear once every two clock cycles. So we can improve the up counter to count 2 bits (bit '1' and an SN bit) at a clock cycle, which will further reduce the computing latency. And since one of these two bits is determined, we don't need to use the FSM-based SNG to generate it. The N-bit

FSM could be shrunk to $N-1$-bit. Note that as we count 2 bits at a cycle, the up counter of the FSM will increase by 2 at each cycle, which is shown in Fig. 2.8(b).

To further improve the hardware performance of the Scaled-CBSC multiplication approach, we try to optimize the CBSC kernel upon the original design under the case of the maximum number of the scaling bits. By observing the 3rd and the 4th line in Fig. 2.8(a), we notice that the output value of FSM in the SNG is just a reversal series of the output value of the down counter before the SC multiplication terminates. This means that when the FSM output value equals to the initial value of the down counter, the whole counting process will be ended. So the down counter is not required anymore, we simply remove it from the CBSC structure, thus further reduce the area and power of our proposed scaled CBSC multiplication. We show the improved scaled CBSC multiplication structure in Fig. 2.9. Note that since we don't need to count the first $2^N/2$ cycles and count 2 bits at each cycle now, the counting process should not be stopped when $FSM < \lfloor (w-2^N/2)/2 \rfloor$, which is equivalent to $FSM < w[N-2:1]$. For the odd number case, we need to count an extra "half" cycle. In this case, as w is an odd number, $w[0]=1$. We just need to add one to the final output result. Finally, to combine the odd and even cases together, the output result will be added by $x[N-1] \cdot w[0]$ when the process stops. And since $x[N-1]=1$, $x[N-1] \cdot w[0] = w[0]$. We show the optimization in Fig. 2.9.

2.4 Experimental Results and Discussions

In this section, we evaluate the performance of the proposed counter-based stochastic-behaving approximate integer multiplier (COSAIM) and scaled counting-based stochastic

Table 2.1: Hardware performance comparison of Scaled-CBSC with other state of art works.

8-bit						
	M	**Area /μm^2**	**Delay /ns**	**ADP**	**Power /μW**	**Energy /pJ**
CBSC [40]	0	487.96	215.04	104930	26.85	13.75
Improved CBSC[6]	0	657.72	130.56	85873	39.04	10.00
Scaled-CBSC	1	460.51	176.64	81344	22.60	11.57
	2	483.38	176.64	85385	24.15	12.36
	3	575.13	63.36	36440	32.50	4.16
COSAIM	0	660.77	2.05	1355	36.20	0.14
Scaled-COSAIM	1	692.29	2.07	1433	37.57	0.15
	2	710.33	2.07	1470	38.34	0.15
	3	707.79	2.07	1465	39.33	0.16
16-bit						
COSAIM	0	2575.50	3.85	9916	163.95	0.66
Scaled-COSAIM	1	2622.00	3.87	10472	165.33	0.66
	2	2691.13	3.88	10144	157.34	0.63
	3	2700.53	3.87	10216	157.86	0.63
	4	2655.55	3.88	10130	159.86	0.64

computing multiplication approach, named *Scaled-CBSC* multiplication under 8-bit and 16-bit precision. We also compare the proposed approach against the original CBSC (counting-based stochastic computing) multiplication baseline [40], and other state of art works with the same precision.

2.4.1 Experimental Setup

To evaluate the performance of the proposed Scaled-CBSC multiplication and COSAIM design, we first compare the error metrics and the hardware performance of Scaled-CBSC and COSAIM with the original baseline version: the CBSC multiplication proposed by Sim [40]. We also compare Scaled-CBSC multiplication with a state of art work: an optimized CBSC multiplication [6]. For the 16-bit Scaled-CBSC and COSAIM multiplication, as CBSC multiplication will take 2^{15} clock cycles in average to finish the multiplication, which is too long and doesn't make sense. We just demonstrate the results of COSAIM and Scaled-COSAIM designs, which only require a single clock cycle to finish the process.

All the aforementioned multipliers are implemented in Verilog HDL and synthesized with Synopsys Design Compiler using EDK 32nm standard cell library [11] as single-cycle designs. For fair comparison, the power and energy consumption of all the aforementioned multipliers are measured under the same working frequency (250MHz).

For the error metrics evaluation, we developed behavioral simulation models for the listed multipliers with all the possible number of scaling bits in MATLAB and measured the accuracy using 1 million random inputs uniformly distributed over the set $\{0, 1, ..., (2^N - 1)\}$, $(N = 8, 16)$. The errors are reported with respect to the exact results. The error metrics used to report the error behavior includes: mean error (mean of the absolute value of the error) and standard deviation. Note that both of the improved versions of the CBSC method and the proposed COSAIM design only do improvements on the computing latency and the hardware performance (area/power/energy), and the error metrics keeps the same as the CBSC baseline. Thus for the error metrics, we just compare our proposed Scaled-CBSC/COSAIM multiplication with the original CBSC baseline.

2.4.2 Performance Evaluation

We show the error metrics for CBSC and Scaled-CBSC multiplication with 8-bit and 16-bit in Fig. 2.10 and Fig. 2.11, respectively. Note that the original CBSC method

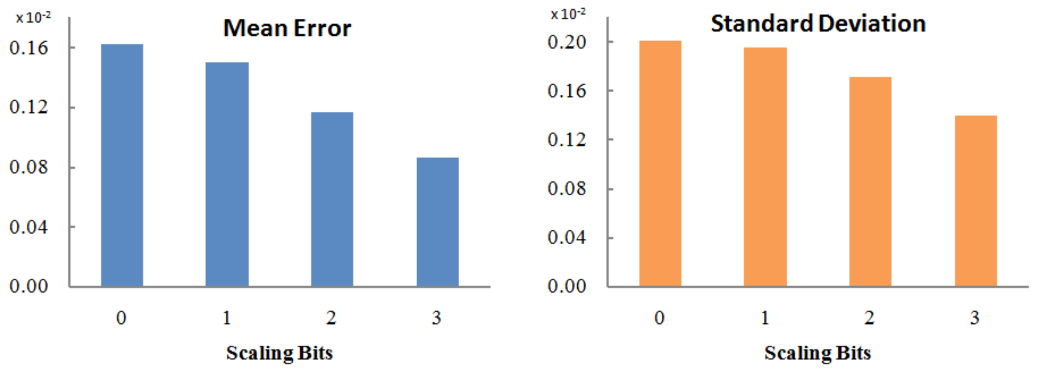


Figure 2.10: The mean error and standard deviation of 8-bit Scaled- CBSC multiplication approach.

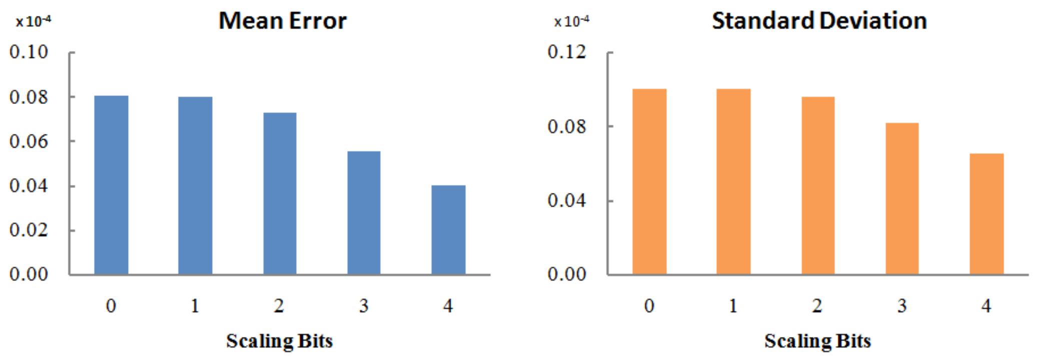


Figure 2.11: The mean error and standard deviation of 16-bit Scaled- CBSC multiplication approach.

can be treated as Scaled-CBSC with '0' scaling bit, which is shown at the leftmost location in Fig. 2.10 and Fig. 2.11.

Fig. 2.10 shows that with 3 scaling bits, the Scaled-CBSC multiplication can improve the mean error and the standard deviation upon the CBSC baseline by up to 46.6% and 30.4%. Fig. 2.11 shows that with 4 scaling bits, Scaled-CBSC multiplication can improve the mean error and the standard deviation upon the CBSC baseline by up to 50.3% and 34.9%. Furthermore, from Fig. 2.12, we can see that with scaling bits, Scaled-CBSC

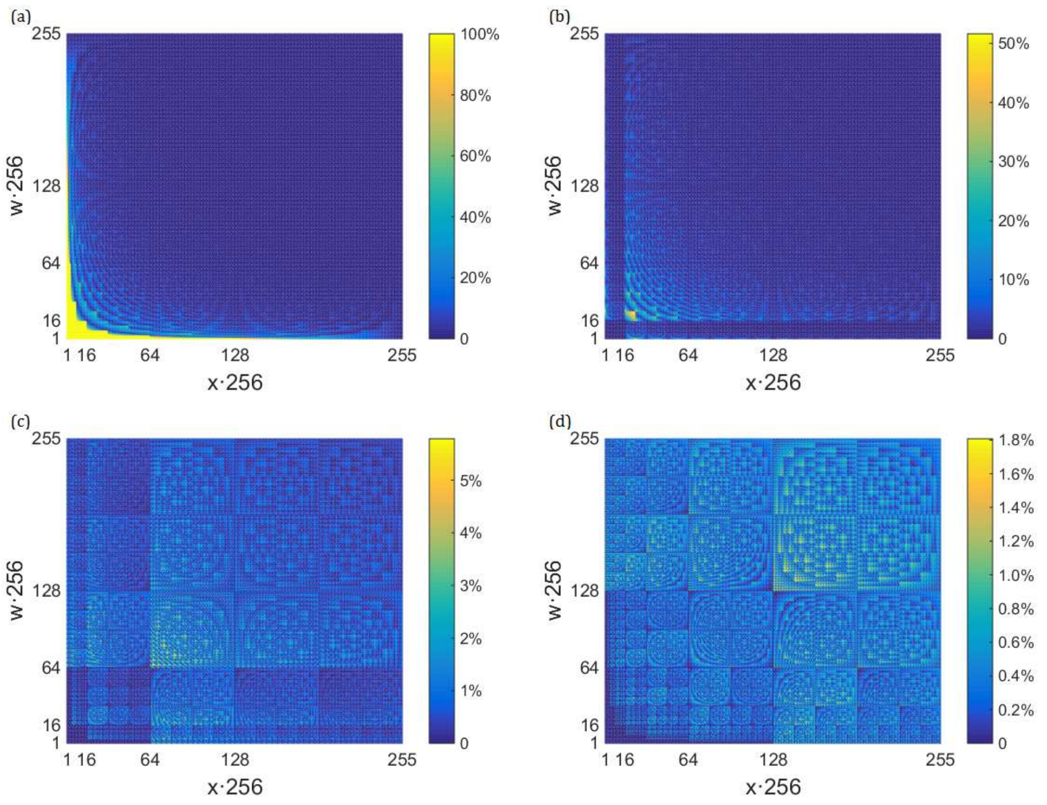


Figure 2.12: The absolute relative error distribution of Scaled-CBSC with different number of scaled bits: (a) $M = 0$; (b) $M = 1$; (c) $M = 2$; (d) $M = 3$.

can improve the relative error profile of the original CBSC significantly. With 1, 2, 3 scaling bits, Scaled-CBSC can reduce the maximum relative error from 100% to 51.6%, 5.8% and 1.8%, respectively.

We demonstrate the hardware performance of the Scaled-CBSC multiplication and COSAIM design, comparing them with the CBSC baseline and a state of art work in Table 2.1. M is the number of scaling bits we used in the multiplication. By observing Table 2.1, we can see that with 8-bit precision, Scaled-CBSC can improve the delay upon the CBSC baseline by up to 70.5% with 17.9% area overheads, and improve the ADP (area-delay product) and energy consumption upon the CBSC baseline by up to 65.3% and 69.7%, respectively; and improve all the four metrics (area, delay, ADP, energy) upon the state of art work by 12.6%, 51.5%, 57.6% and 58.4%, respectively, with 3 scaling bits. For 8-bit COSAIM design, though it will introduce some area overheads, it can significantly improve the delay, ADP and total energy consumption by 99.0%, 98.7% and 99.0%, respectively.

2.4.3 An Image Processing Application Evaluation

Now, we show how the proposed Scaled-CBSC multiplication approach compare to state of art methods in a multimedia application. Discrete cosine transformation (DCT) is a commonly used lossy image compression method. The quality of the compressed images is usually evaluated using metrics such as PSNR (peak signal noise ratio) and higher PSNR value represents better image quality. We implement the proposed Scaled-CBSC multiplication approach with different number of scaling bits in the DCT-iDCT (inverse DCT) workloads, and compare with the CBSC baseline on five example images, considering both 8-bit and 16-bit precision. As mentioned before, $M = 0$ represents the CBSC baseline. We show the results of image compression in Table 2.2. For 8-bit precision, we can see that 1 or 2 scaling bits will not improve the image quality much. With 3 scaling bits, Scaled-CBSC can improve the image quality of 5.9dB in average upon the CBSC baseline. For 16-bit precision, 1 scaling bit can improve the image quality of 16.8dB in average, and 2 scaling bits are enough to achieve the best image quality improvement. Here "INF" in Table 2.2 represents "infinity large", which means the final output image is completely the same as the input with no quality loss. To make a complementary, COSAIM design will not be demonstated due to the same error behavior as the baseline.

Table 2.2: PSNR (dB) for images after DCT-iDCT using CBSC multiplications.

8-bit						
M	**Lena**	**Boat**	**Barbara**	**House**	**Pepper**	**Avg. (dB)**
0	32.24	31.57	32.70	31.95	32.62	32.22
1	32.43	31.77	32.93	32.17	32.93	32.45
2	32.47	31.85	33.04	32.32	33.03	32.54
3	38.17	37.92	38.58	37.80	37.97	38.09
16-bit						
M	**Lena**	**Boat**	**Barbara**	**House**	**Pepper**	**Avg. (dB)**
0	78.23	76.08	77.54	77.32	74.99	76.83
1	INF	91.52	96.30	93.29	93.29	93.60
2	INF	INF	INF	INF	INF	INF
3	INF	INF	INF	INF	INF	INF
4	INF	INF	INF	INF	INF	INF

2.5 Summary

The work in this chapter proposes a new counter-based stochastic-behaving approximate integer unsigned multiplier, *COSAIM* for many emerging machine learning hardware implementation. *COSAIM* is an accelerated design of recently proposed counter-based stochastic multiplier. As a result, it still keeps the advantages of stochastic computing such as built-in progress reconfigurability for progressive performance-accuracy trade-off. The work in this chapter also proposes a novel scaled counting-based stochastic computing multiplication design, named *Scaled-CBSC*. The proposed design introduces scaling bits to promise the inputs of the SC multiplication kernel to be larger than 0.5, avoiding the "small" number region which leads to large relative error of SC multiplication. Numerical results shows that 8-bit *Scaled-CBSC* and Scaled COSAIM designs with 3 scaling bits can achieve up to 46.6% and 30.4% improvements in mean error and standard deviation, respectively; reduce the peak relative error from 100% to 1.8%. For hardware performance, the *Scaled-CBSC* can improve 12.6%, 51.5%, 57.6%, 58.4% in delay, area, area-delay product, energy consumption, respectively, over the state of art work. Though COSAIM will introduce some resource overhead, it can significantly reduce the delay and the area-delay product. For discrete cosine transformation (DCT) application, 3 scaling bits are required for 8-bit *Scaled-CBSC* and COSAIM multiplication to significantly improves the image quality by 5.9dB.

Chapter 3

A Counting-Based Stochastic Computing Division

3.1 Review of Correlation Based SC Division

Correlated division, or CORDIV, proposed by Te-Hsuan [5] explored the fact that division could be computed as the conditional probability of two SN numbers: $P_{X_1|X_2}$. The formula can be simplified to

$$P_{X_1|X_2} = p_{x_1,x_2}/p_{x_2} = x_{1,SN}/x_{2,SN} \tag{3.1}$$

when x_1 and x_2 are highly correlated, where p_{x_2} is the event of x_2 and p_{x_1,x_2} is the joint event of x_1 and x_2. As a result, the output probability $P_{Quotient}$ can be expressed by the ratio of number of bit '1' in the dividend and the divisor SN bit-streams: $N^1_{Dividend}/N^1_{Divisor}$. Thus, CORDIV requires that the divisor is always larger than the dividend to avoid the quotient being larger than one. This work proved that with highly correlated input SN bit-streams, CORDIV could obtain much more accurate results when compared with Gaines' design [10]. But the accuracy of CORDIV method depends on the quality of input SN bit-streams. The framework of CORDIV design is shown in Fig. 3.1.

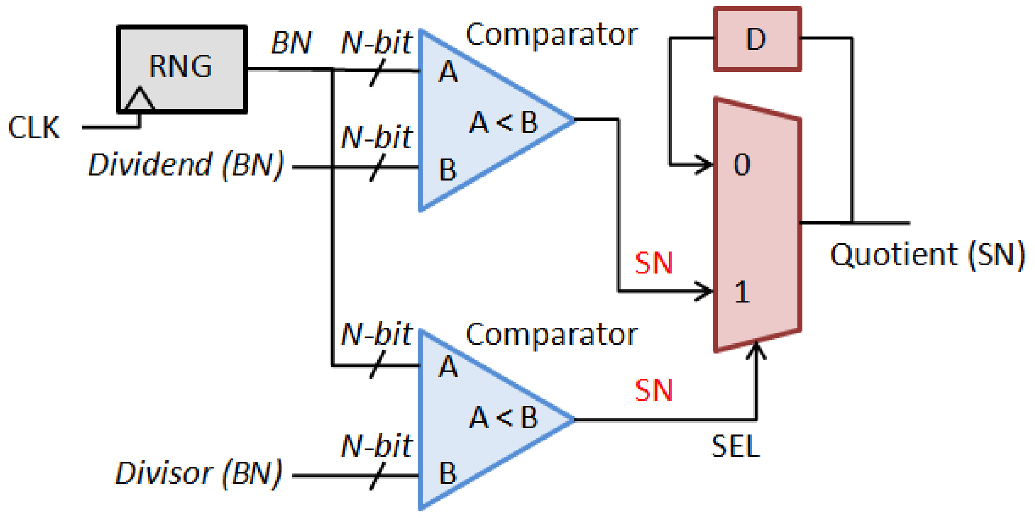


Figure 3.1: CORDIV design diagram.

Specially designed SNGs shown in Fig. 3.1 [5] has been proved that could mitigate this problem. However, the accuracy still depends on the distribution of bit '1' in input SN bit-streams, shown in Fig. 3.2. Adding skewed synchronizer could improve accuracy, but was still not accurate enough, especially in certain output value ranges [51].

Based on the correlated division method and the aforementioned SNG design in the CBSC-MUL framework (Sec. 2.1), a counting-based SC division, or *CBDIV*, is proposed to improve the latency and accuracy of SC division in this chapter.

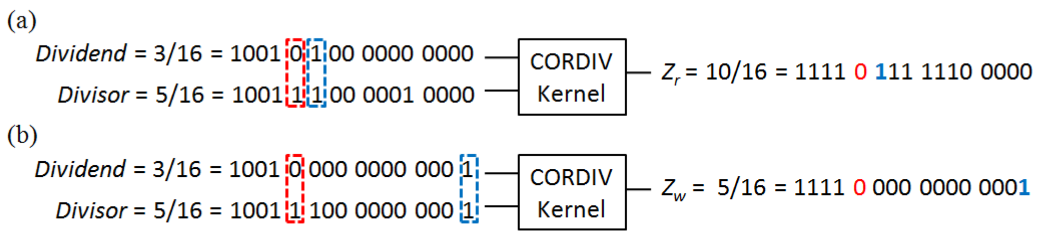

(a)
Dividend = 3/16 = 1001 0 1 00 0000 0000
Divisor = 5/16 = 1001 1 1 00 0001 0000
CORDIV Kernel
Zr = 10/16 = 1111 0 111 1110 0000
(b)
Dividend = 3/16 = 1001 0 000 0000 000 1
Divisor = 5/16 = 1001 1 100 0000 000 1
CORDIV Kernel
Zw = 5/16 = 1111 0 000 0000 0001

Figure 3.2: CORDIV method input SN bit-streams with different bit distribution: (a) Quotient result with appropriate bit distribution. (b) Quotient result with extreme bit distribution.

3.2 Proposed Counting-Based SC Division Design

In this section, we try to mitigate the aforementioned issues in the existing correlated division designs and propose a new SC division method based on a more efficient counting-based SC (CBSC) concept.

From Fig. 3.2, we can see that the location of bit ‘1’ in the divisor may significantly affect the quotient accuracy. The two input probabilities are 3/16 and 5/16, respectively, whose exact quotient result would be 0.6. The first 1-0 pair and the following 1-1 pair in the divisor and the dividend SN bit-streams are in red and blue blocks, respectively. And the output in the quotient bit-stream at the corresponding location, which are the first bit ‘0’ and following ‘1’ are also marked with red and blue, respectively. As CORDIV method will consistently output bit ‘0’ if the divisor and the dividend SN have 1-0 pair followed by 0-0 pairs at the corresponding location until a 1-1 pair appears. And will consistently output ‘1’ if the two input bit-streams have a 1-1 pair followed by 0-0 pairs until a 1-0 pair appears. Suppose the input SNs have appropriate bit ‘1’ distribution as shown in the upper case in Fig. 3.2, CORDIV computes $Z_r = 10/16 = 0.625$. But if we have extreme bit ‘1’ distribution given as the lower case in Fig. 3.2, too many 0-0 pairs appear between 1-0 pair and the following 1-1 pair, CORDIV computes $Z_w = 5/16 = 0.3125$, which is far from the exact result.

To solve this problem, we propose a new division method called CBDIV, which stands for *Counting-Based Division*. Unlike CORDIV [5] or ISCBDIV [51], we don't require a synchronizer or expensive RNG-comparator structure to guarantee a strong correlation of two input SN bit-streams. Instead, the SN generation in CBDIV utilizes a deterministic pattern shown in Fig. 2.1(a), which is isolated from the random fluctuations.

Since we use the deterministic pattern to generate the divisor SN bit-stream, the location of bit '1' is fixed. We can easily arrange the dividend SN bit-stream to favor the 1-1, 0-0 pairs as many and fast as possible, as the location of bit '1' in the dividend SN bit-stream is completely determined according to the divisor SN. That means if a bit in divisor SN is '1', the bit in dividend SN at the corresponding location will also be set to '1' to guarantee all the 1-1 pairs to be continuously found. So, before $S_{Dividend}$ has used up all bit '1's in the SN bit-stream, $S_{Dividend}$ is completely the same as $S_{Divisor}$. In other words, when the first 1-0 pair occurs, meaning that the dividend (which is less than divisor by design) has already used up all the bit '1' and no more 1-1 pair is left, which can be illustrated in an example shown in the first 2 rows in Fig. 3.3(a). Here, the value of the dividend SN $S_{Dividend}$ and the divisor SN $S_{Divisor}$ are 6/16 and 9/16, respectively.

As we can see, after the arrangement, the right parts of $S_{Divisor}$ and $S_{Dividend}$ in the blue dash line block in Fig. 3.3(a) are exactly the same. Then based on CORDIV method [5], the quotient SN bit-stream $S_{Quotient}$ simply obtains all its bit '1's in the right half at the 3rd row in Fig. 3.3(a). A counter which increases by 1 in every clock cycle is

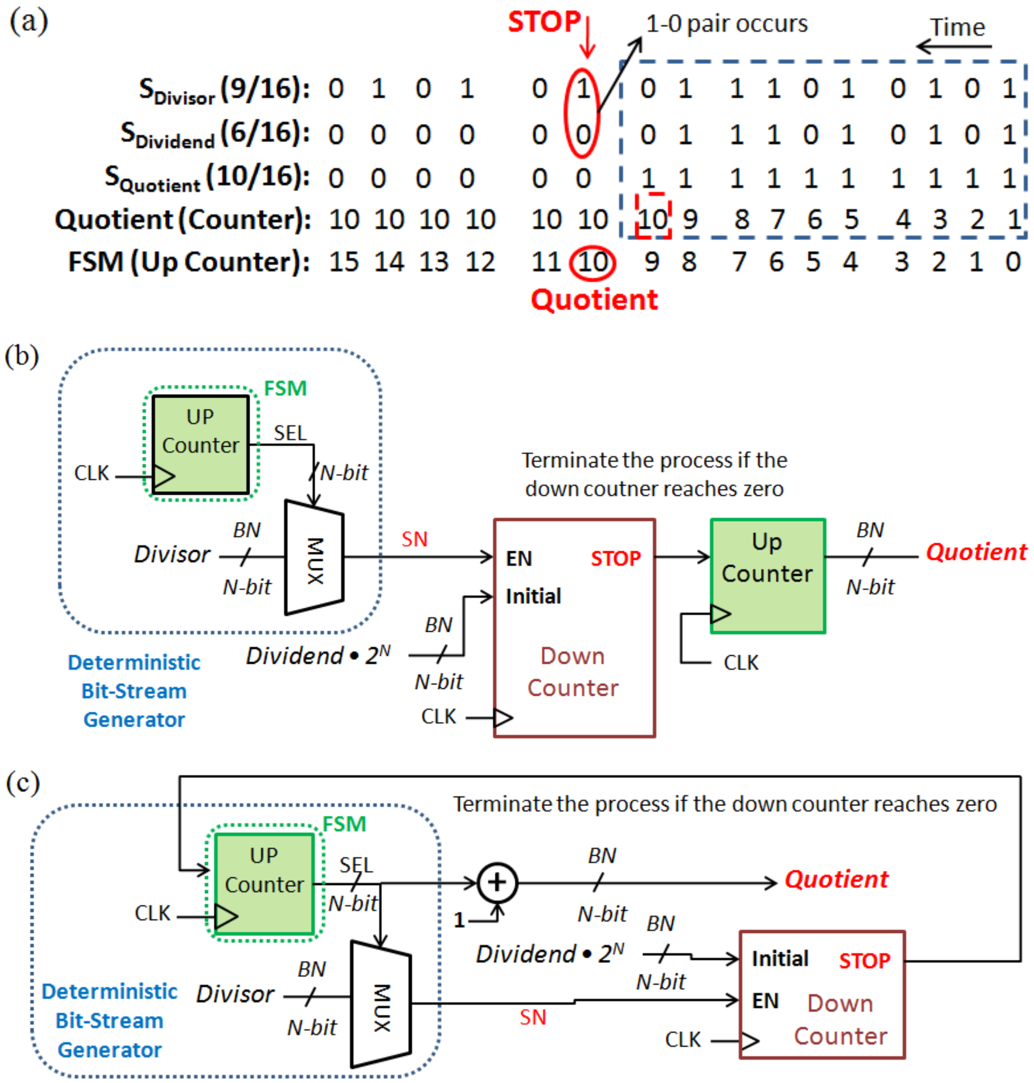

(a)
STOP
1-0 pair occurs
Time
S_Divisor (9/16): 0 1 0 1 0 1 0 1 1 1 0 1 0 1 0 1
S_Dividend (6/16): 0 0 0 0 0 0 0 1 1 1 0 1 0 1 0 1
S_Quotient (10/16): 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
Quotient (Counter): 10 10 10 10 10 10 10 9 8 7 6 5 4 3 2 1
FSM (Up Counter): 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Quotient
(b)
FSM
UP Counter
SEL
N-bit
CLK
Divisor
BN
N-bit
MUX
SN
Deterministic Bit-Stream Generator
Dividend • 2^N
Terminate the process if the down coutner reaches zero
EN
Initial
STOP
Down Counter
Up Counter
Quotient
(c)
Terminate the process if the down counter reaches zero
1
Initial
EN
Down Counter

Figure 3.3: (a) The proposed CBDIV method. (b) The proposed CBDIV design. (c) The optimized CBDIV design.

enough to get the binary form value, which is shown in the 4th row in Fig. 3.3(a). We notice that the remaining bits outside the blue frame in $S_{Quotient}$ are all bit '0'. As a result, we don't need to count anymore and can simply terminate the process, thus reduce the computing time. The result now counts $10/16 = 0.625$, which is a good approximation to the exact result $2/3 = 0.667$.

Based on our CBDIV method, the CBDIV design actually requires 3 counters to realize the process, 2 up counters and 1 down counter, as shown in Fig. 3.3(b). One up counter is used as an FSM (finite state machine) to generate the divisor SN bit-stream. The bit-stream follows similar low discrepancy distribution proposed in [40]. Because the two input bit-streams, $S_{Dividend}$ and $S_{Divisor}$, are completely the same before the division process ends as we have discussed before. We only need to generate one of them. Here, we only generate $S_{Divisor}$. The other up counter is used to convert the quotient SN bit-stream $S_{Quotient}$ to binary form. It simply increases by one in every clock cycle since the bits in $S_{Quotient}$ are always '1' before the division process ends. The down counter, with an initial value of $Dividend \cdot 2^N$, determines when to terminate the process. The enable signal "EN" of the down counter is connected to the $S_{Divisor}$ signal. So, the down counter will decrease by one if bit '1' appears in $S_{Divisor}$.

By observing the 4th and 5th row in Fig. 3.3(a), we notice that the FSM output is always one smaller than the output quotient value before the process terminates, which means we can infer the quotient from the FSM output value even without a counter. So we combine the 2 up counters together. Considering the $S_{Divisor}$ generation, we keep the up counter used as the FSM. When the division process finishes, we simply add one to the output of the FSM to obtain the quotient. Now we only require 2 counters to form our CBDIV design. The optimized CBDIV design is shown in Fig. 3.3(c).

Table 3.1: Hardware performance for SC and fixed-point dividers.

Approaches	Area (μm^2)	Critical Path (ns)	Avg. Latency (cycles)	Avg. Delay (ns)	ADP	Power (µW)	Avg. Energy (pJ)
Fixed-Point	1090	3.16	17	53.72	58555	41.4	7.038
DFSM-DIV [45]	520	1.24	128	158.72	82534	22.3	28.544
CORDIV [5]	381	0.94	128	120.32	45842	14.7	18.765
ISCBDIV [51]	394	0.98	128	125.44	49423	15.8	20.211
CBDIV (proposed)	**299**	**1.72**	**44**	**75.68**	**22628**	**10.9**	**4.796**

3.3 Experimental Results and Discussions

In this section, we evaluate the performance of the proposed *counting-based SC divider*, CBDIV. We also compare CBDIV against the binary logic baseline and the state-of-the-art works.

3.3.1 Experimental Setup

To evaluate the performance of the proposed CBDIV design, we compare the error behavior and the hardware performance of CBDIV with a fixed-point divider using long division algorithm. The area, the critical path, the average latency (clock cycles), the average delay, the area and delay product (ADP), the power and the total energy consumption are measured to evaluate the hardware performance of CBDIV. We also compare CBDIV with several state of art works such as DFSM-DIV (*SC division with deterministic finite state machines*) [45], CORDIV [5], and ISCBDIV [51]. All the approaches are with binary number inputs of 7-bit precision (corresponding to 128-bit length bit-stream).

The dividers are implemented in Verilog and synthesized with Synopsys Design Compiler using EDK 32nm standard cell library [11]. For fair comparison, all the dividers are binary in and binary out, SNGs which generate SN bit-streams and a counter which counts the value of the result SN bit-stream have been added to the in-stream SC-based division designs. The hardware performance comparison of the dividers are shown in Table 3.1. For fair power and energy consumption comparison, the input clock frequency of all the dividers are set to be 100MHz. In addition, energy consumption is measured based on the total execution time and its power consumed during the whole division process.

The error behavior is measured using *root mean square error* (RMSE). To evaluate the error behavior, we develop behavioral simulation models for all the SC-based dividers listed in Table 3.1 in MATLAB and measure the accuracy of over 100, 000 operations. To promise the dividend be smaller than the divisor, we generate the dividend in the range of [0, 0.5], and the divisor in the range of [0.5, 1].

3.3.2 Hardware Performance

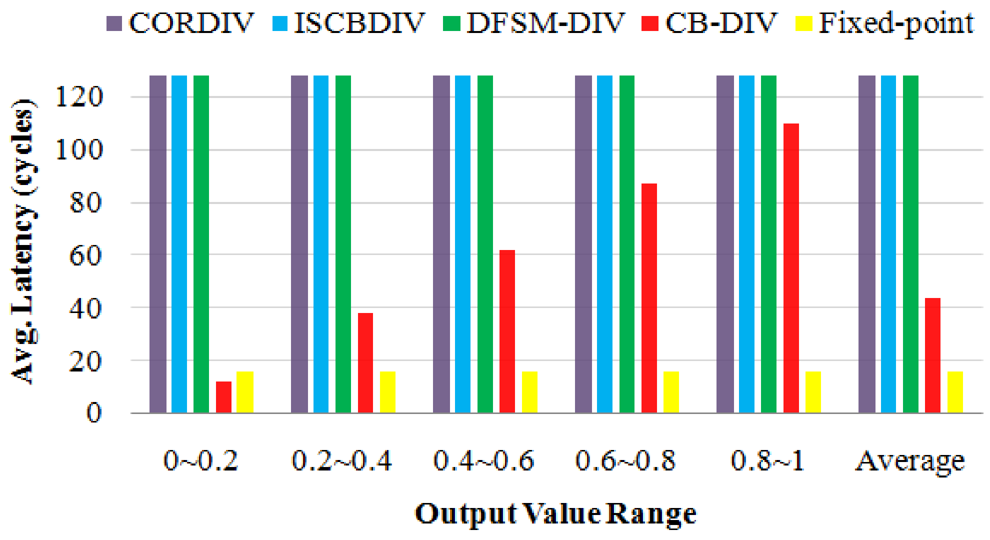
From the second column in Table 3.1, we observe that among all the listed stochastic dividers, our proposed CBDIV is the most area efficient, doing 72.6% of area reduction when compared to the fixed-point divider (binary logic) baseline, and outperforms state of art stochastic dividers by at least 21.5%. As the latency of CBDIV design is determined by the value of the dividend, which is proved in Sec. 3.2, the latency of CBDIV design will vary according to the inputs. This is different from the latency of previous SC division designs when certain precision is determined, which is a fixed value. We show this property in Fig. 3.4. So we use *Avg. Latency* (average latency) and *Avg. Delay* to measure the average computing time of CBDIV and compare these two values against previous works. From column 4, we can see that CBDIV saves 65.6% of latency in average; and from column 5, the *Avg. Delay* of CBDIV is 75.68ns, outperforms state of art works by at least 37.1%. Also, though the latency of CBDIV is 40.9% larger than which of the fixed-point divider, the area and delay product (ADP), is much smaller than the fixed-point divider (61.4% smaller), and improves by at least 50.6% when compared to previous works as shown in column 6.

From the last two columns in Table 3.1, we observe that CBDIV performs the best both in power and energy consumption among all the listed division designs. In power aspect, CBDIV outperforms state of art works by 25.9%. For the energy consumption aspect, we note that in-stream SC-based division designs will cost even more energy than the fixed-point divider when SNGs and the counter are taken into consideration. CBDIV design has solved this problem by improving the computing latency, doing 31.9% in energy reduction when compared to the fixed-point divider baseline.

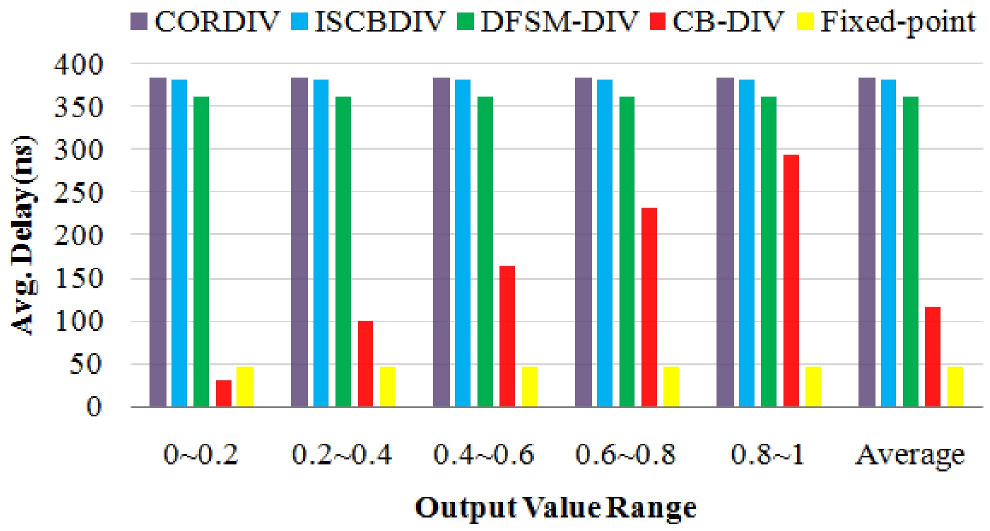
3.3.3 Error Behavior

Fig. 3.5 shows the accuracy comparison among the proposed CBDIV method with the baseline fixed-point divider and existing approaches: CORDIV [5], ISCBDIV [51], DFSM-DIV [45]. LFSR (linear feedback shift register) is utilized to generate SN bit-streams for binary inputs in state of art approaches for comparison.

We show the results over five non-overlapped output value ranges as well as the average RMSE value obtained from 100, 000 computations for all the algorithms. From Fig. 3.5, we can observe that CBDIV outperforms state of art works in all of the five non-overlapped output value ranges. Compared with DFSM-DIV, which has the smallest RMSE value in average, CBDIV makes an improvement by 77.8%. To make a supplement, note in Fig. 3.5, CBDIV shows similar RMSE values in different output value ranges, different



(a)



(b)

Figure 3.4: Latency and delay comparison among CBDIV and other dividers with different output value ranges (7-bit precision): (a) Average latency. (b) Average delay of the whole process.

from performances of the existing works which is due to the intrinsic property of different SC division methods.

Furthermore, we show the error behavior of CBDIV with different bit-stream length (precision) compared with the fixed-point baseline and state of art works in Fig. 3.6. In Fig. 3.6, we show that even with 5-bit precision, the average value of RMSE of CBDIV is still 15.4% better than DFSM-DIV with 7-bit precision.

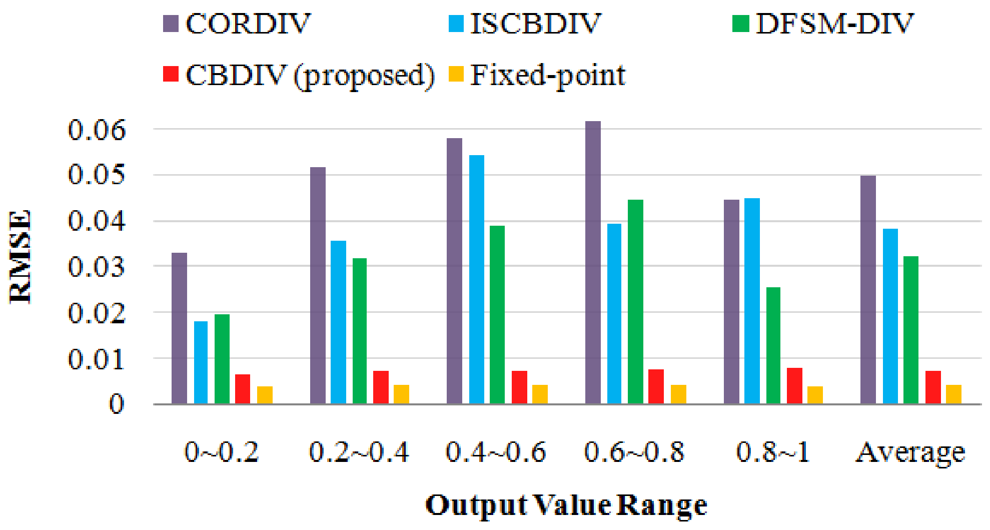


Figure 3.5: Error behavior of CBDIV and other dividers with different output value ranges (7-bit precision).

3.3.4 Comparison in an Image Processing Application

We implement the proposed CBDIV method for an image process application, *Contrast Stretch* to further compare the performance against the baseline design and the state of art works.

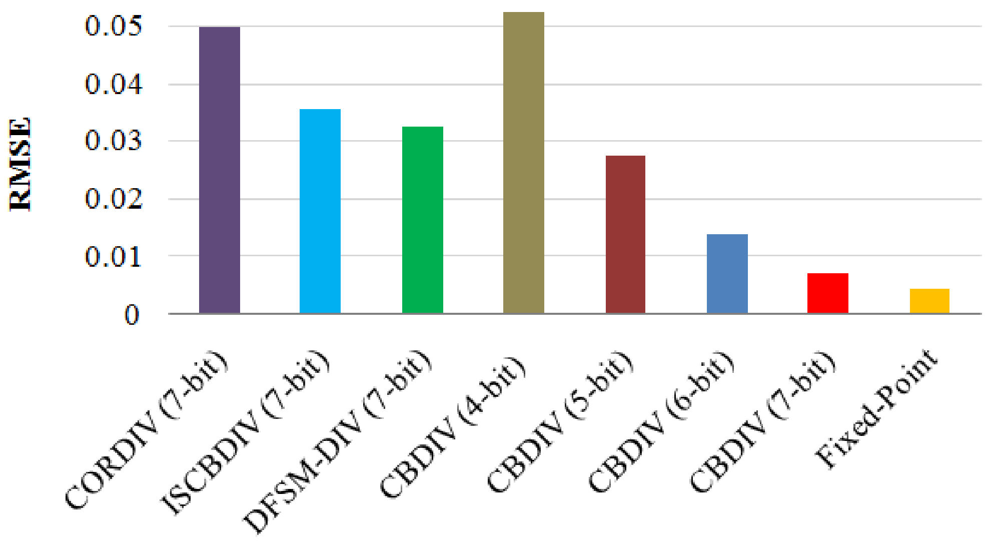


Figure 3.6: Error behavior of CBDIV with different precision.

The bit-stream length of CBDIV, ISCBDIV [51] and DFSM-DIV [45] implemented are all 256-bit, as the pixels in the input figure with JPEG format are 8-bit precision. The contrast-stretched pixel $G(x, y)$ of an arbitrary input image pixel $I(x, y)$ can be calculated as (3.2).

$$G(x, y) = \frac{I(x, y) - I_{min}}{I_{max} - I_{min}} \cdot 255 \tag{3.2}$$

Where I_{max} and I_{min} are the maximum and minimum pixel value in the image $I(x, y)$, respectively. Note that we first perform the division of $\frac{I(x,y)-I_{min}}{I_{max}-I_{min}}$, since SC dividers always require the dividend smaller than the divisor. We also utilize the CBSC method to do the multiplication illustrated in Sec. 3.1. 255 can be treated as $255/256 \cdot 2^8$, then w in CBSC multiplier will be 255/256 and the process requires 255 clock cycles to finish. According to the deterministic pattern used in CBSC method, all bit '1's in the divisor $(\frac{I(x,y)-I_{min}}{I_{max}-I_{min}})$ bit-stream will appear in these 255 clock cycles, meaning that we only need to count how many bit '1's are there in the divisor bit-stream. For CBDIV, the counting process is finished at the same time when CBDIV terminates. We don't require any more action and simply output the CBDIV result as the value of $G(x, y)$. While the outputs for ISCBDIV and DFSM-DIV are SN bit-streams, so we just add a counter to obtain $G(x, y)$.

We apply contrast stretching to several input images, using PSNR (Peak Signal Noise Ratio) to evaluate the output image quality. Contrast stretched images with an exact divider are used as the baseline. The PSNR values of all the tested images with different SC dividers are shown in Table 3.2. We can observe that the proposed CBDIV method outperforms the two state of art SC dividers in all the tested images, and improves the image quality by 20.6 dB in average.

Table 3.2: PSNR for contrast stretch application using SC dividers

PSNR (dB)			
	ISCBDIV	DFSM-DIV	CBDIV
Map	34.84	36.28	**53.69**
Portrait	36.19	27.30	**57.40**
Woman	30.22	26.37	**53.11**
Lena	32.80	25.35	**54.62**

3.4 Summary

The work in this chapter proposes a fast and energy efficient approximate divider design based on stochastic computing division design, CBDIV, which exploits both the correlation requirement of existing SC-based division methods and the high efficiency of counting-based SC scheme. CBDIV fits well with the hybrid computing in which binary and SC implementations are both required for overall better application performance, like image processing. Experimental results show that the proposed CBDIV outperforms state of art works by 77.8% in accuracy, 37.1% in delay, 21.5% in area, 50.6% in ADP and 25.9% in power. CBDIV also saves 31.9% in energy consumption when compared to the fixed-point division baseline, and is much more energy efficient than early proposed SC-based dividers if the inputs and outputs are both binary numbers. Furthermore, CBDIV with 5-bit precision can even outperform state of art works with 7-bit precision in accuracy by 15.4%. Finally, we compare CBDIV with other state of art SC dividers in contrast stretch application and shows that CBDIV can improve the accuracy with 20.6dB in average, which is a huge improvement.

Chapter 4

HEALM: Hardware Efficient Approximate Logarithmic Multiplier With Reduced Error

4.1 Approximate Logarithmic Multiplier

The classic approximate logarithmic multiplier was proposed by Mitchell, also called *ALM* [31]. It shows good overall performance and has flexibility for trade-offs among area, power and accuracy. Specifically, for ALM design, the two inputs A and B are first represented by the following format: $2^{k_a} \cdot (1 + x)$ and $2^{k_b} \cdot (1 + y)$, respectively. Then the multiplication result can be approximated as (4.1).

$$C_{ALM} = \begin{cases} 2^{k_a+k_b} \cdot (1 + x + y), & x + y < 1, \\ 2^{k_a+k_b+1} \cdot (x + y), & x + y \geq 1 \end{cases} \tag{4.1}$$

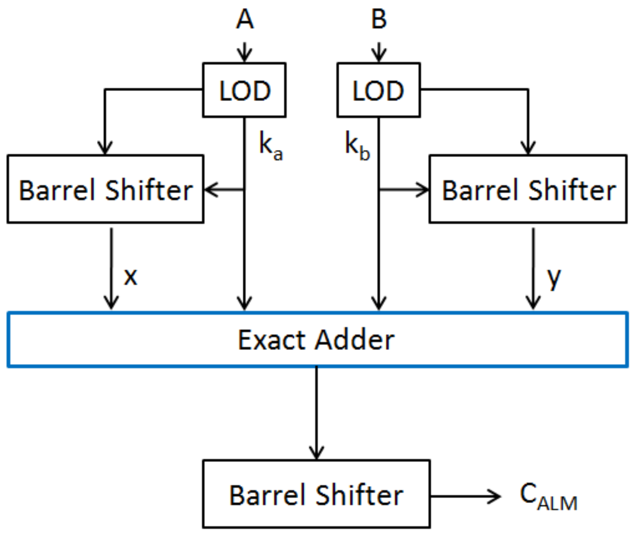


Figure 4.1: The design of approximate log-based multiplier.

Here C_{ALM} is the approximate multiplication result. The ALM design requires four steps to finish the multiplication process. The architecture of ALM design is shown in Fig. 4.1.

First it utilizes leading-one detectors (LOD) to find the leading bit '1' as the integer part; second, barrel shifters are used to re-align the rest of the bits as the fraction part; then it sums the two fraction and integer parts up as $k_a + k_b + x + y$; and finally it shifts back with the same bits. Although ALM suffers from high absolute MRED (mean relative error distance) and peak relative error of 3.76% and 11.11%, respectively, it can perform a good trade-off among accuracy, area and power.

This chapter will focus on the 8-bit and 16-bit precision hardware efficient approximate logarithmic multiplier design and demonstrate the superior performance of the proposed new design against the ALM [31] baseline and other state of art works like LeAp [9], REALM [39], ALM-SOA [30], ILM-EA [3] and ILM-AA [3].

4.2 Proposed Hardware-Efficient Approximate Logarithmic Multiplier

In this section, we show the details of the proposed hardware-efficient ALM design by considering the bit or width truncation in the mantissa summation part and error compensation at the same time.

Consider an inexact adder with N-bit inputs A and B, then the sum S has $N+1$ bits. Let k be the number of bits in the lower part of the sum which are approximated. The binary representation of A is $A_{N-1}A_{N-2}...A_kA_{k-1}...A_0$ and B is in a similar form. The upper part $A_{N-1}A_{N-2}...A_k$ and $B_{N-1}B_{N-2}...B_k$, which are denoted as A_H and B_H, respectively, will perform the exact summation to obtain the higher part of S, $S_NS_{N-1}...S_k$, which is denoted as S_H. The lower part $A_{k-1}...A_0$ and $B_{k-1}...B_0$, which are denoted as A_L and B_L, will perform the approximate summation to obtain the lower part of S: $C_{k-1}S_{k-1}...S_0$. Note that C_{k-1} is the carry bit to the exact summation of upper parts.

We implement two representative approximate adders (or inexact adders), one is the truncation adder (TA), the other is the set one adder (SOA) [30] with error improvement to the ALM. These adders have very small complexity, which are suitable for implementing our *HEALM* designs, named as *HEALM-TA* and *HEALM-SOA* with error improvement. To carry out the error compensation, we first analyze the error profile of ALM when the exact adder used for the mantissa summation is replaced with an approximate adder. Then, we perform specified error compensation for HEALM with each approximate adder under different k values, where k represents the number of bits in the inexact mantissa summation part, to achieve the best trade-off among the error metrics and the hardware resources. The structure of the *HEALM* design is shown in Fig. 4.2.

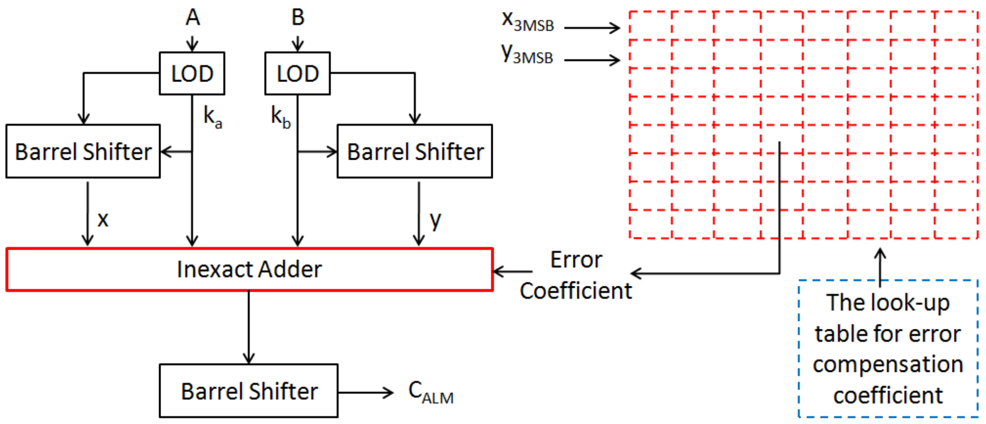
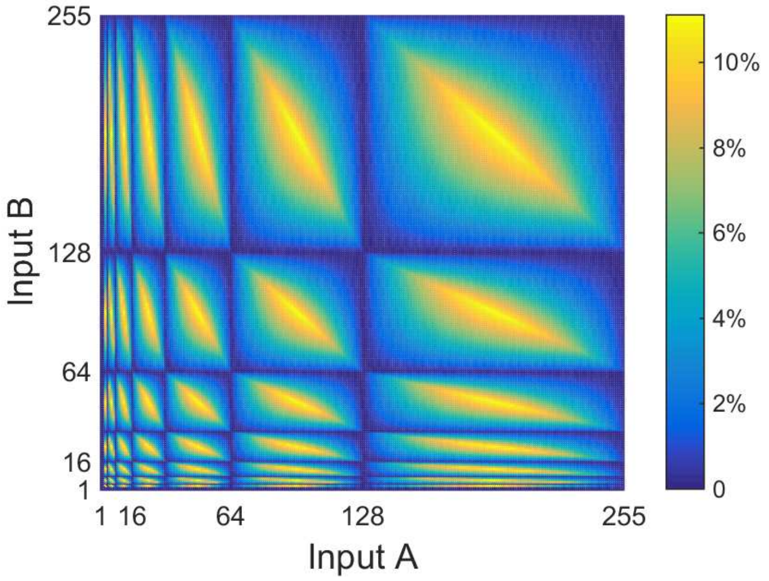


Figure 4.2: HEALM design: mantissa summation in ALM design replaced with an error compensated approximate adder.

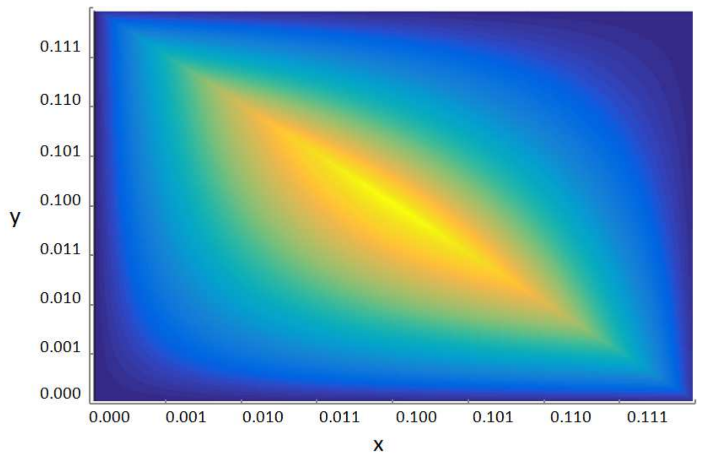
4.2.1 HEALM With Truncation Adder: HEALM-TA

The Error Behavior of ALM-TA

Before we discuss our proposed HEALM design with truncation adder, or *HEALM-TA*, we need to show the error behavior of ALM with a simple TA, represented as *ALM-TA*. First, we give a brief introduction on the concept of TA. The TA simply truncates the lower part of the inputs A and B, which makes the inexact part of the mantissa summation S_L equal to zero. Thus the mantissa summation with TA will only calculate the upper part S_H. An N-bit adder is actually truncated to an $N-k$-bit adder. We use a 7-bit case with $k=4$ to describe the concept of TA, which is shown in Fig. 4.4(a).

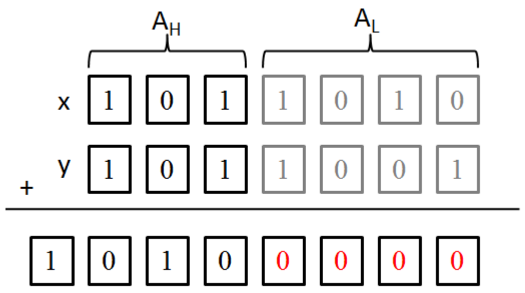


(a)

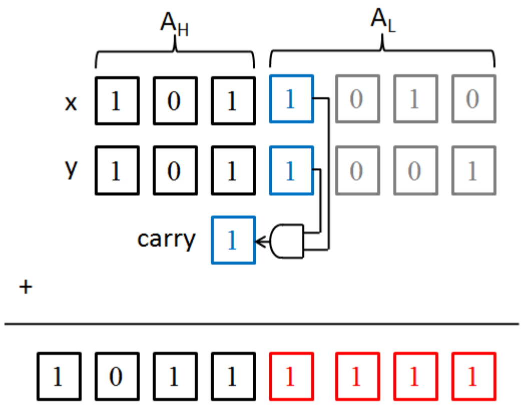


(b)

Figure 4.3: (a) The error behavior of ALM (8×8). (b) The error behavior of the fractional part of ALM in an power-of-two interval.



(a)



(b)

Figure 4.4: (a) A 7-bit truncation adder (TA) example. (b) A 7-bit set one adder (SOA) example.

The error behavior of ALM-TA is similar as the error behavior of the ALM method. We have shown the ALM method in (4.1) in Sec. 4.1, and now the exact multiplication written in log-based form can be expressed as (4.2).

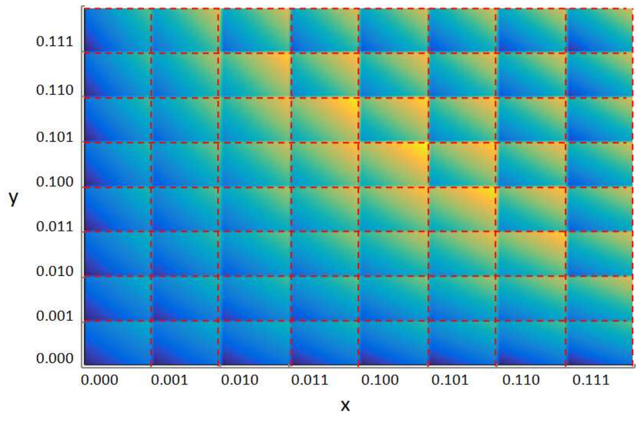
$$\begin{aligned} C_{Exact} &= 2^{k_a+k_b} \cdot (1+x) \cdot (1+y) \\ &= 2^{k_a+k_b} \cdot (1+x+y+xy) \end{aligned} \tag{4.2}$$

$$\begin{aligned} Error_{ALM} &= C_{Exact} - C_{ALM} \\ &= \begin{cases} 2^{k_a+k_b} \cdot (xy), & x+y<1, \\ 2^{k_a+k_b+1} \cdot (1-x-y+xy), & x+y \geq 1 \end{cases} \end{aligned} \tag{4.3}$$

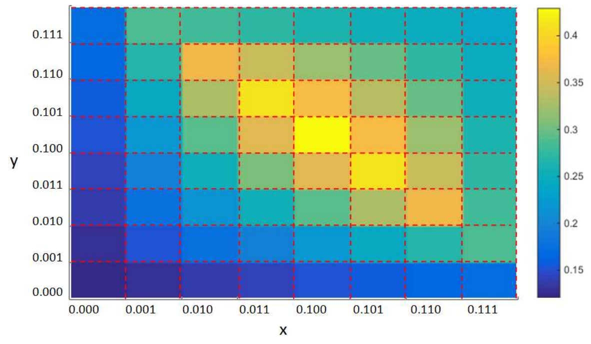
$$
\begin{aligned}
Error_{ALM-TA} &= C_{Exact} - C_{ALM-TA} \\
&= \begin{cases}
2^{k_a+k_b} \cdot (1 + x_H x_L + y_H y_L \\
+x \cdot y - 1 - x_H - y_H), & x + y < 1, \\
2^{k_a+k_b+1} \cdot (1 + x_H x_L + y_H y_L \\
+x \cdot y - 2 \cdot x_H - 2 \cdot y_H), & x + y \geq 1
\end{cases} \\
&= \begin{cases}
2^{k_a+k_b} \cdot (x_L + y_L + xy), & x + y < 1, \\
2^{k_a+k_b+1} \cdot (1 + x_L + y_L \\
+x \cdot y - x_H - y_H), & x + y \geq 1
\end{cases}
\end{aligned}
\tag{4.4}
$$

Based on (4.1) and (4.2), we can calculate the error of ALM as (4.3). The error behavior of ALM showing a proportional replication in each power-of-two interval is demonstrated in Fig. 4.3. Hence, we can perform error compensation to the fractional part before barrel shifting operation to save resource, which is also demonstrated in [9].

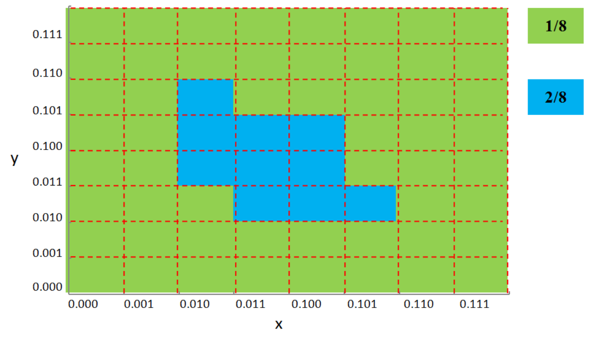
After replacing the exact mantissa summation in ALM ("$x+y$") with approximate summation by using TA, the error will also accumulate. We use $Error_{ALM-TA}$ to represent it from time being. The error behavior of ALM-TA can be calculated as (4.4) and also has a proportional replication in each power-of-two interval. Fig. 4.5(a) demonstrates the error behavior of ALM-TA with $k = 4$ in an interval. Notice that the error behavior though distributes nearly symmetric, which is similar as the error profile of ALM in a single interval. It further shows proportional replication in each 1/8 interval. Also, for TA summation, no matter what the lower half of the inputs are (here represented as x_L and y_L, as the inputs of



(a)

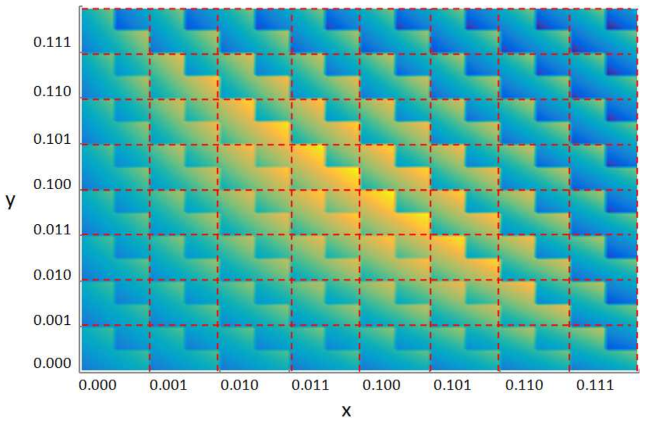


(b)

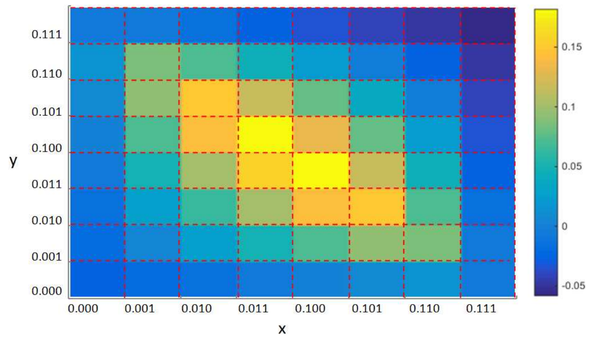


(c)

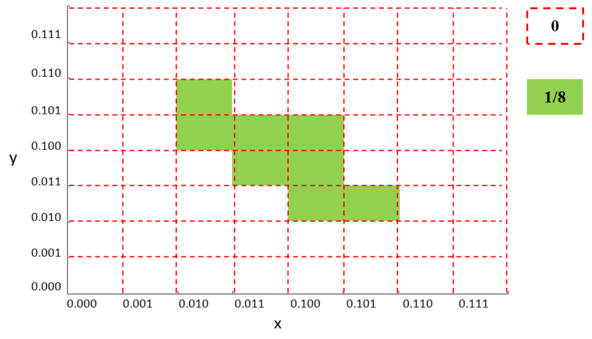
Figure 4.5: (a) The error behavior of ALM-TA with $k = 4$. (b) The error behavior of ALM-TA with $k = 4$ doing average operation in each block. (c) The error compensation pattern for HEALM-TA with $k = 4$.



(a)



(b)



(c)

Figure 4.6: (a) The error behavior of ALM-SOA with k=4. (b) The error behavior of ALM-SOA doing average operation in each block. (c) The error compensation pattern for HEALM-SOA with k=4.

the mantissa summation is the fractional part x and y of input A and B, respectively), the approximate summation is only determined by the exact summation part $(x_H + y_H)$. Thus, we partition the fractional xy-space into 64 blocks (8×8) with the red dash line, which are the most significant 3 bits (3 MSBs) of x and y, as shown in Fig. 4.5; and recalculate the average error in each block as shown in Fig. 4.5(b). Also, for ALM-TA with more than 3 bits in the exact summation part ($k < 4$), we still use the 8×8 blocks partition to calculate the average error to save resource. The experimental results shown in Sec. 4.3 will prove that the partition with 8×8 is sufficient to achieve acceptable accuracy improvement and good resource saving.

The Proposed HEALM-TA Error Compensation

Based on the aforementioned observation on the error behavior of ALM-TA, we can perform specified error compensation and propose our HEALM idea. We first generate a lookup table, which is of the same size as 8×8 blocks partition, as an error compensation pattern. An example of the pattern is shown in Fig. 4.5(c). The error coefficient, Err_{coeff}, which is added to the approximate mantissa summation, is generated by searching the lookup table based on the 3 MSBs of x and y to perform the specified error compensation. Note that when the error compensation pattern is simple (usually the value of k is large), such as the example we show in Fig. 4.5(c), the lookup table can be simplified to several large squarish area. Like the case shown in Fig. 4.5(c), the blue area is equivalent to the sum of 3 rectangular regions, which can be described much simpler than an 8×8 lookup table, thus saving the resource consumption.

$$C_{HEALM-TA} = \begin{cases} 2^{k_a+k_b} \cdot (1 + x_H + y_H \\ +Err_{coeff}), & x + y < 1, \\ 2^{k_a+k_b+1} \cdot (x_H + y_H \\ +Err_{coeff}), & x + y \geq 1 \end{cases} \tag{4.5}$$

The HEALM-TA method can be expressed as (4.5), where $C_{HEALM-TA}$ is the product of HEALM-TA method. And the value of the error coefficients are determined by the average error of each block. We notice that if $x+y \geq 1$, the error coefficient Err_{coeff} will be added twice. Thus, the equivalent error in these blocks where $x + y \geq 1$ should be as half as its initial value. So we divided the Err_{coeff} for these blocks to half. And for those blocks where $x+y$ could be either smaller or larger than 1, we further perform error compensation arrangement to achieve the possible smallest peak relative error. Note that the mantissa summation of $x + y$ is replaced with the approximate summation now, so we need to do quatization of the error coefficients to ensure that the precision of these coefficients no larger than the precision of the exact summation part. In the case of $k = 4$ as shown in Fig. 4.5(c), the exact summation only has 3 bits. So Err_{coeff} also need to be a 3-bit parameter. Actually in this case, the error coefficient will either be 1/8 or 2/8 as shown in Fig. 4.5(c). Our proposed HEALM-TA design performs well especially when the value of k is large. And 8-bit HEALM-TA with $k = 3$ can improve the traditional ALM design in both error metrics and area, which is never achieved by the previous works with 8-bit precision. We'll prove this later in Sec. 4.3.

Single Coefficient Mode: HEALM-TA-S

Furthermore, we propose a single coefficient mode, named as *HEALM-TA-S* to perform error compensation on ALM-TA with almost no resource overheads. As an N-bit simple TA with k bits truncated, it consists of 1 HA (half adder) and $N-k-1$ FAs (full adder), which is shown in Fig. 4.7(a) (in the example of $k = 4$, the exact summation part includes 2 FA and 1 HA). To perform the simplest error compensation, the error coefficient

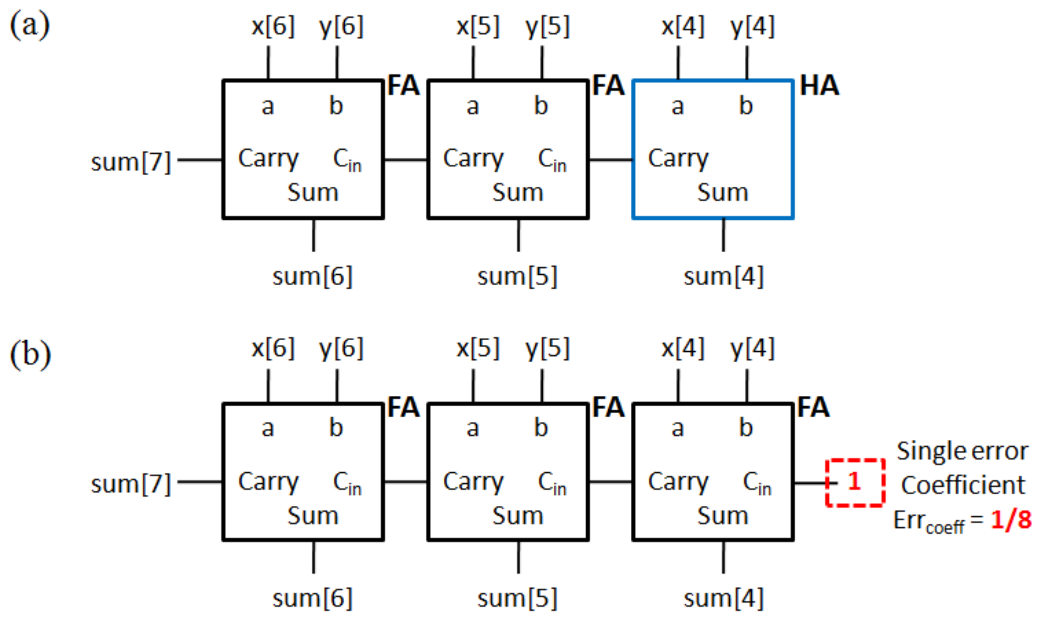


Figure 4.7: The mantissa summation unit of HEALM-TA-S design with $k = 4$: (a) The exact summation part with no error compensation. (b) The exact summation part improved with a single error coefficient, replacing the HA with an FA.

for the whole fractional space is set to be the same value, which is $2^{-(N-k)}$ (1/8 in this case); and the HA is replaced with an FA at the LSB (least significant bit) location to obtain the smallest resource overheads. The structure of the mantissa summation part of HEALM-TA-S design is shown in Fig. 4.7(b). Note that the input carry bit (C_{in}) for the

FA at LSB is always set to ‘1’ according to the error coefficient. We’ll prove later in Sec. 4.3 that HEALM-TA-S can perform a good trade-off among the error metrics and the hardware performance especially when k is large for HEALM-TA-S design.

4.2.2 HEALM With Set One Adder: HEALM-SOA

Besides HEALM-TA, we also propose another HEALM design with set one adder, or *SOA*, called *HEALM-SOA*. Simple ALM design with mantissa summation replaced with SOA (ALM-SOA) has already been proposed before [30]. Based on ALM-SOA, we further perform error compensation similar to Sec. 4.2.1.

In an SOA, different from TA, all the bits in S_L part are set to logic ‘1’ to produce a balanced error in the ALM-SOA method. For the S_H, which is the exact summation part, $S_H = A_H + B_H + C_{in}$, where the carry bit C_{in} is obtained by doing an AND operation of the MSB in A_H and B_H (A[k-1] and B[k-1], respectively), as expressed in (4.6), suppose the SOA is an N-bit summation with k bits in the approximate summation part S_L.

$$\begin{aligned} S_H = S[N:k] &= A[N-1:k] + B[N-1:k] + C_{in} \\ C_{in} &= A[k-1]B[k-1] \\ S_L[i] &= 1, i \in [0, k-1] \end{aligned} \tag{4.6}$$

Then, based on (4.1) and (4.6), we can calculate the error of ALM-SOA in an power-of-2 interval as (4.7), where $C_{in} = x[k-1]y[k-1]$, and $S_{SOA} = \Sigma 2^i/2^N$, $i \in \{0, 1, ..., k-1\}$. Similar as ALM-TA, we show the error behavior of ALM-SOA ($k = 4$) with an example in Fig. 4.6(a).

$$
\begin{aligned}
Error_{ALM-SOA} &= C_{Exact} - C_{ALM-SOA} \\
&= \begin{cases}
2^{k_a+k_b} \cdot (1 + x_H x_L + y_H y_L + x \cdot y & \\
-1 - x_H - y_H - C_{in} - S_{SOA}), & x + y < 1, \\
2^{k_a+k_b+1} \cdot (1 + x_H x_L + y_H y_L + x \cdot y & \\
-2 \cdot (x_H + y_H + C_{in} + S_{SOA})), & x + y \geq 1
\end{cases} \\
&= \begin{cases}
2^{k_a+k_b} \cdot (x_L + y_L + x \cdot y - C_{in} - S_{SOA}), & x + y < 1, \\
2^{k_a+k_b} \cdot (1 + x_L + y_L + x \cdot y & \\
-x_H - y_H - C_{in} - S_{SOA}), & x + y \geq 1
\end{cases}
\end{aligned}
\tag{4.7}
$$

The HEALM-SOA idea is similar as HEALM-TA. The error compensation pattern of HEALM-SOA is shown in Fig. 4.6(c). We partition the fractional space into 8×8 blocks and calculate the average error following the same way as HEALM-TA, which is shown in Fig. 4.6(b). Then based on the error distribution of ALM-SOA, we generate a specified error compensation pattern in a lookup table form. The error coefficient which is added to the mantissa summation part is determined by the 3 MSBs of x and y as HEALM-TA. Similar to HEALM-TA, HEALM-SOA also selects the error compensation patterns to achieve the smallest possible peak relative error and can provide improvement upon the traditional ALM design in terms of both the error metrics and resource consumption. We'll show this later in Sec. 4.3.

Note that unlike HEALM-TA, the LSB summation of the exact summation part in HEALM-SOA should consider the carry bit C_{in} from the S_L (approximate summation part). The LSB summation also requires a FA instead of HA in the exact summation part of HEALM-SOA. We cannot directly add a bit '1' as error compensation to the LSB location. So HEALM-SOA will not have a single error coefficient mode like "HEALM-SOA-S".

4.3 Experimental Results and Discussions

In this section, we evaluate the performance of the proposed *hardware-efficient approximate logarithmic multiplier with reduced error*, named HEALM under 8-bit precision. We also compare HEALM against the ALM (approximate logarithmic multiplier) baseline [31] and other state of art works with the same precision. Furthermore, we demonstrate 16-bit HEALM design results compared with the baseline and state of art works as a complementary.

4.3.1 Experimental Setup

To evaluate the performance of the proposed HEALM design, we first compare the error metrics and the hardware performance of HEALM with its original version: a classical ALM proposed by Mitchell, which is selected as the baseline. We also compare HEALM with other state of art improved ALMs. These improved ALMs include: LeAp [9], REALM [39], ALM-SOA [30], ILM-EA [3], ILM-AA [3]. For REALM design, we compare REALM8 which did the same partition in the fractional space (in an power-of-2 interval) as HEALM does for fair comparison.

All the above mentioned 8-bit multipliers are implemented in Verilog HDL and synthesized with Synopsys Design Compiler using EDK 32nm standard cell library [11] as single-cycle designs, and at the same timing constraints of 2.5ns (400 MHz working frequency) for area and power consumption comparison. For 16-bit multipliers, we implemented with the same library but at the timing constraints of 5ns (200MHz).

For the error metrics evaluation, we developed behavioral simulation models for all the multipliers listed in Table 4.1 in MATLAB and measured the accuracy using 1 million random inputs uniformly distributed over the set $\{0, 1, ..., (2^8 - 1)\}$. The errors are reported with respect to the exact results. The error metrics used to report the error behavior include: mean error (mean of absolute relative error, also referred as MRED in some previous works [3]); and peak error (maximum value of the absolute relative error). All the error metrics are in percentages.

4.3.2 Performance Evaluation

The error metrics and the hardware performance for the implemented multipliers are shown in Table 4.1. Since the proposed HEALM designs utilize the inexact adder in the mantissa summation, we use a parameter k in Table 4.1 to represent the number of bits in the inexact summation part. For example, in the demonstrated cases shown in Fig. 4.4(a) and Fig. 4.4(b), k equals to 4. For ALM, LeAp, and ILM-EA designs, as these multipliers do not have an inexact summation unit, k equals to 0. For the REALM design, the value of the error configuration parameter mentioned in the work [39] is equivalent to the number of bits in the inexact summation part, which is represented by k in our work. To avoid ambiguity, we use the same notation as HEALM design does for easy comparison.

Table 4.1: Error metrics and hardware performance comparison for the 8-bit multipliers.

Logarithmic Multiplier Design	**k**	**Mean Error /%**	**Peak Error /%**	**Area /μm^2**	**Power /μW**
HEALM-TA (proposed)	1	1.12	4.86	1216.84	114.50
	2	1.40	8.25	938.05	92.17
	3	2.17	9.75	743.63	74.14
	4	3.66	13.77	595.46	51.41
HEALM-TA-S (proposed)	1	3.21	11.11	763.70	71.19
	2	3.17	12.02	716.69	65.68
	3	3.39	13.79	614.01	56.62
	4	4.26	17.12	534.72	42.32
HEALM-SOA (proposed)	1	1.13	4.71	1175.41	113.96
	2	1.38	5.90	966.76	90.32
	3	1.78	7.65	808.94	75.05
	4	3.12	12.17	664.33	59.55
ALM with/without Approximate Adder					
ALM [31]	0	3.76	11.11	820.63	72.83
ALM-TA	1	4.02	12.03	763.45	69.41
	2	4.79	13.83	702.71	65.07
	3	6.58	17.25	612.74	56.20
	4	10.29	23.53	533.19	41.73
ALM-SOA [30]	1	3.50	11.11	776.92	72.11
	2	3.23	11.46	736.26	68.54
	3	3.07	12.36	702.96	64.46
	4	3.47	14.13	596.98	50.63
Other Improved Logarithmic Multipliers from the Literature					
LeAp [9]	0	1.38	4.71	1040.21	106.43
REALM [39]	0	0.90	3.96	1235.14	124.75
	1	1.06	4.76	1128.40	108.48
	2	1.81	8.27	993.96	96.46
	3	3.25	12.80	908.82	82.56
	4	6.58	23.07	709.57	62.40
ILM-EA [3]	0	2.84	11.11	1221.92	107.89
ILM-AA [3]	1	2.91	12.25	1186.85	104.72
	2	3.09	14.24	1046.06	95.03
	3	3.64	17.35	985.57	87.74
	4	5.47	23.47	887.47	77.34

Table 4.2: Error metrics and hardware performance comparison for the 16-bit multipliers.

Approach	**k**	**Mean Error /%**	**Peak Error /%**	**Area /μm^2**	**Power /μW**
ALM [31]	0	3.76	11.11	1825.52	110.91
REALM [39]	0	0.75	3.70	2383.36	164.50
	9	1.06	5.27	1572.90	94.07
LeAp [9]	0	0.98	4.76	1990.71	128.20
ALM-TA	9	4.88	12.93	1263.86	66.26
HEALM-TA-S	**9**	**4.87**	**12.02**	**1267.16**	**66.45**
HEALM-TA	**9**	**1.64**	**5.83**	**1511.39**	**87.62**
ALM-SOA [30]	9	3.07	12.03	1383.56	74.10
HEALM-SOA	**9**	**1.38**	**5.15**	**1577.47**	**91.89**

Table 4.1 demonstrates that with the same value of k, the proposed HEALM-TA design improves the error metrics upon the ALM-TA design, reducing up to 6.63%, 9.76%, in mean and peak error, respectively; HEALM-SOA improves the error metrics upon the ALM-SOA design, reducing up to 2.37%, 6.40%, in mean and peak error, respectively. When compared with the ALM baseline, HEALM-TA and HEALM-SOA can improve the mean / peak error by 2.64% / 6.25%, and 2.63% / 6.40%, respectively. When compared with REALM, which is the state of art work, the HEALM designs can improve mean error, peak error, area and power consumption by up to 2.92%, 9.3%, 16.08%, 17.61% respectively with the same value of k. The smallest mean error that HEALM-TA and HEALM-SOA can achieve are 1.12% and 1.13%, respectively. And the smallest peak error that HEALM-TA and HEALM-SOA can obtain are 4.86% and 4.71%, respectively. The 16-bit multiplication results are summarized in Table 4.2. Due to limited space, we simply show the results of $k = 9$ and compare with several state of art works. 16-bit HEALM-TA can improve all of the four design metrics (mean error, peak error, area, power) against the ALM baseline by 2.12%, 5.28%, 17.21%, 21.00%; and 16-bit HEALM-SOA can improve the design metrics by 2.38%, 5.96%, 13.59%, 17.15%.

Considering the trade-off among error metrics improvement and resource consumption, the previous works can improve either error metrics or resource consumption (area, power) aspect, but can hardly improve both of these aspects especially when the precision is small (like 8-bit precision) as shown in Table 4.1. To better illustrate this, we show the relationship between the mean error / peak error and area / power for all the listed multipliers in Fig. 4.8. The rectangular area with the red dash border line in all four sub

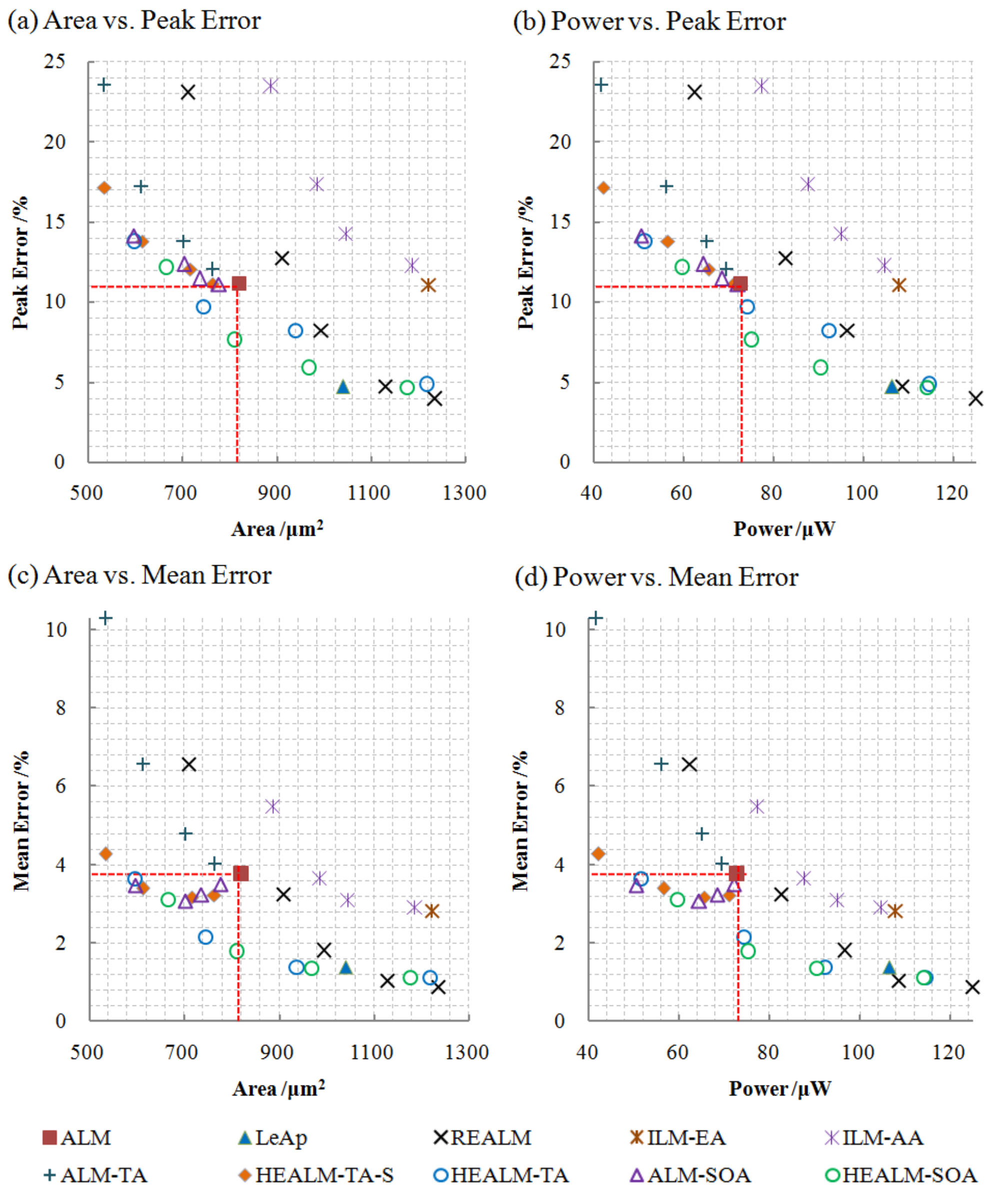


Figure 4.8: Comparison of HEALM with baseline and state of art works.

figures represents that a design outperforms the classical ALM design both in error metrics and resource consumption aspects. Notice in Fig. 4.8(a), only the proposed HEALM-TA and HEALM-SOA with $k = 3$ improve both the peak error and area aspects, decreasing the peak error with 1.36% and 3.46%, respectively. In Fig. 4.8(c), though some previous works like ALM-SOA outperforms ALM in both mean error and area aspects, only a little improvement in mean error (at most 0.69%) was obtained. In contrast, the proposed HEALM-TA and HEALM-SOA with $k = 3$ reduce the mean error by 1.59% and 1.98% when compared to ALM, respectively; and provides 9.38% and 1.42% in area reduction at the same time. Besides, HEALM-TA and HEALM-SOA design with $k = 4$ can reduce the mean error by 0.10% and 0.64% when compared to ALM and reduce power by 29.41% and 18.23% at the same time.

The results of HEALM-TA-S (single error coefficient mode) design in Table 4.1 shows that HEALM-TA-S can do better trade-offs between accuracy and resource consumption especially when the value of k is large. In case of $k = 4$, which is the largest value of k, HEALM-TA-S decreases the mean / peak error by up to 6.03% / 6.11%, respectively when compared to ALM-TA. Note that HEALM-TA-S achieves this improvement with almost no resource overheads. It also saves 34.84% / 41.89% of area / power with 8-bit inputs, respectively; and 30.59% / 40.09% with 16-bit inputs when compared to the ALM baseline.

4.3.3 An Image Processing Application Evaluation

Now, we show how the proposed *HEALM* designs compare to state of art methods in an multimedia application. Discrete cosine transformation (DCT) is a commonly used lossy image compression method. The quality of the compressed images is usually evaluated using metrics such as PSNR (peak signal noise ratio) and higher PSNR value represents better image quality. We implement the proposed HEALM design with 8-bit precision in the DCT-iDCT (inverse DCT) workloads, and compare with other logarithmic multipliers

Table 4.3: PSNR (dB) for images after DCT-iDCT using logarithmic multipliers.

Approach	**Lena**	**Boat**	**Barbara**	**House**	**Pepper**
ALM (baseline)	19.1	18.7	19.3	18.4	18.7
k = 1					
ILM-AA	27.9	26.6	28.1	24.8	27.4
ALM-TA	18.9	18.6	19.2	18.2	18.5
ALM-SOA	19.2	18.8	19.5	18.6	18.9
REALM	37.2	36.5	36.9	38.4	36.4
HEALM-TA	36.1	35.7	36.2	36.5	35.6
HEALM-TA-S	20.4	19.9	20.6	19.9	20.1
HEALM-SOA	34.3	33.9	34.5	36.2	34.4
k = 2					
ILM-AA	27.6	26.2	27.7	24.3	27.1
ALM-TA	17.4	17.1	17.6	16.5	17.0
ALM-SOA	19.8	19.4	20.1	19.4	19.5
REALM	27.1	26.8	27.1	27.9	27.2
HEALM-TA	33.2	32.7	33.4	35.1	33.3
HEALM-TA-S	19.9	19.5	20.1	19.5	19.6
HEALM-SOA	31.7	31.4	32.3	33.1	31.7
k = 3					
ILM-AA	24.5	23.1	24.5	22.4	23.9
ALM-TA	15.0	14.8	15.2	14.3	14.7
ALM-SOA	19.9	19.5	20.1	19.5	19.6
REALM	21.2	21.2	21.4	19.6	20.8
HEALM-TA	26.5	25.6	26.2	28.1	26.4
HEALM-TA-S	18.8	18.4	18.9	18.7	18.3
HEALM-SOA	29.8	29.5	30.0	31.1	29.7
k = 4					
ILM-AA	20.2	18.9	20.1	18.5	19.5
ALM-TA	14.1	13.7	14.1	13.4	13.7
ALM-SOA	18.9	18.9	19.6	16.7	18.9
REALM	19.8	20.0	19.8	17.8	19.5
HEALM-TA	29.1	28.7	29.1	25.9	28.5
HEALM-TA-S	23.1	22.0	22.6	24.3	22.3
HEALM-SOA	22.2	22.1	22.6	19.4	22.4

on five example images. To be fair, the mantissa summation parts of all the compared logarithmic multipliers are inexact unit, except for the ALM, which is chosen as the baseline. We show the results of image compression in Table 4.3. The result shows that with different values of k, HEALM-TA can improve the image quality upon the ALM baseline by from 7.8$\sim$17.2dB in average and HEALM-SOA can improve 2.9$\sim$15.8dB in average, respectively. Besides, HEALM-TA and HEALM-SOA design outperform all the other state of art works when $k = 2, 3, 4$ by at least 6.3dB, 6.3dB, 8.8dB, respectively. Note that the single coefficient mode design HEALM-TA-S performs the best when $k = 4$, making improvement upon the ALM baseline by 4.1dB in average with extremely low resource consumption as mentioned before. This is due to the error behavior of ALM, whose outputs are always smaller than the exact product. And HEALM-TA-S with $k = 4$ will have a more balanced error than the cases of $k = 1, 2, 3$.

4.4 Summary

The work in this chapter proposes a novel hardware-efficient approximate logarithmic multiplier, called *HEALM*. The proposed design, first determines the truncation width for mantissa summation in ALM. Then the error reduction is performed via a lookup table for multiple partitioned input ranges. Numerical results shows that *HEALM* and its enhanced designs can lead to more accurate results with reduced area and power at the same time than the existing ALM baseline design. It also outperforms the state of art design, REALM, with up to 2.92%, 9.30%, 16.08%, 17.61% improvement in mean error, peak error, area, power consumption for 8-bit precision. For discrete cosine transformation (DCT) application, with different values of k, HEALM-TA can improve the image quality upon the ALM baseline by 7.8$\sim$17.2dB in average and HEALM-SOA could improve 2.9$\sim$15.8dB in average, respectively. Besides, HEALM-TA and HEALM-SOA outperforms all the state of art works with $k = 2, 3, 4$ on the image quality.

Chapter 5

PAALM: Power Density Aware Approximate Logarithmic Multiplier Design

5.1 Power Density Increase in the Approximate Logarithmic Multiplier Design

recent study showed that approximate multiplier designs may lead to unwanted higher power density, thus higher temperature and related reliability issues in the final chip design [52]. The reason is that though approximation schemes reduce both power and area at the same time, the power density, which is power per unit area, can increase as reduction ratios for power and area can be different. It turns out that as shown in Table 5.1, the power densities of recently proposed approximate multipliers are all larger than the fixed point exact multiplier (the baseline) no matter with the same working frequency or with the same throughput. As a result, those approximate multiplier designs can't run on the same frequency and voltage ranges of the original exact multiplier designs under the same temperature constraint [52].

Table 5.1: 8-bit multipliers power density (synthesized using EDK 32nm standard cell library [11])

Approaches	**Area (μm^2)**	**Power Density ($\mu W/\mu m^2$)**	
		same frequency	**same throughput**
FxP (baseline)	**1754.10**	**0.0512**	**0.0512**
ALM [31]	820.63	0.0853	0.0693
LeAp [9]	1040.21	0.1023	0.0831
REALM [39]	1235.14	0.1010	0.0821
ILM-EA [3]	1221.92	0.0883	0.0718
ILM-AA [3]	887.47	0.0871	0.0708
HEALM-TA	595.46	0.0863	0.0702
HEALM-SOA	664.33	0.0896	0.0729

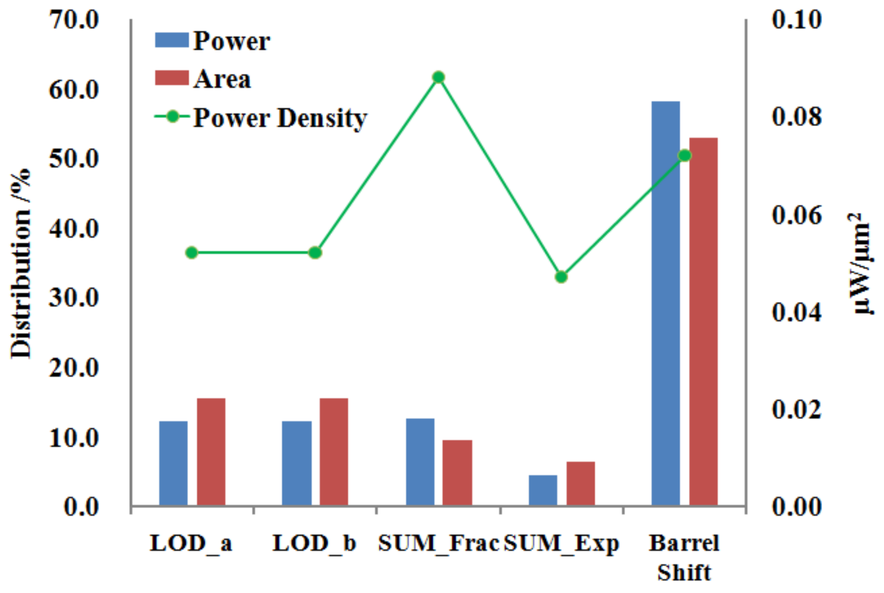
Based on this observation, a power density aware approximate logarithmic multiplier design to achieve superior thermal performance will be demonstrated in this chapter.

5.2 Proposed Power Density Aware ALM

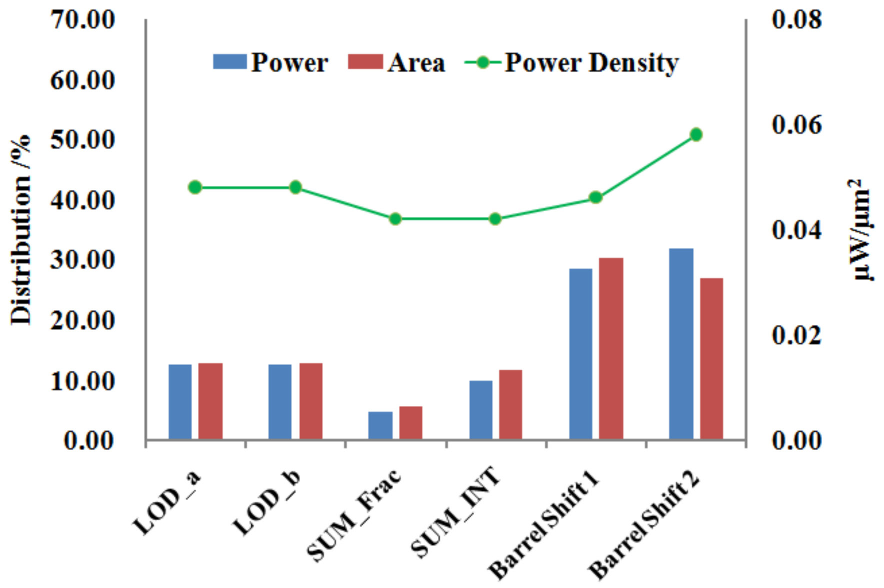
In this section, we present the details of our proposed power density aware logarithmic approximate multiplier (*PAALM*) design to mitigate the increase in the power density of most existing approximate multiplications, especially the log-based multiplication designs.

5.2.1 Power Density Aware Logarithmic Multiplication

The first thing we do is to profile the power consumption of the key components of the ALM design. The goal is to find the local power or power density hot spots so that



(a)



(b)

Figure 5.1: (a) The power, area distribution and power density of each component in the ALM design; (b) The power, area distribution and power density of each component in the PAALM design

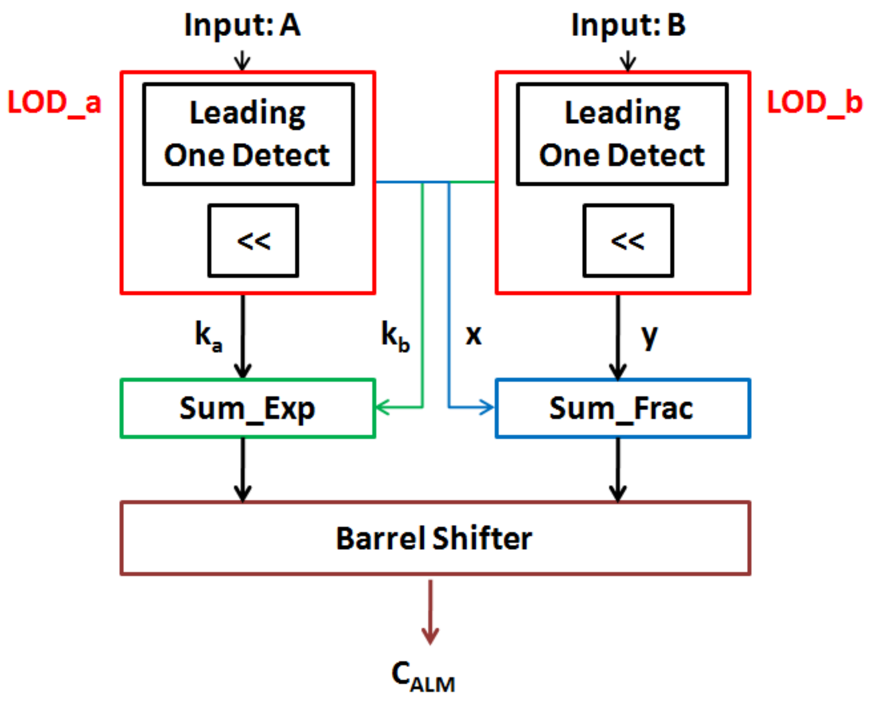


Figure 5.2: The conventional ALM design

we can re-design the multiplier to avoid those hot spots and the overall power density can be reduced in this way.

Specifically, we first show the power, area and the power density for each component in a conventional ALM design in Fig. 5.1(a). In this figure, the *LOD_a* and *LOD_b* modules represent the blocks detecting the leading one bit and performing the bit shifting operation to obtain the exponential part k_a, k_b and the fraction part x, y. The *Sum_Exp* module is used to sum the exponential parts k_a, k_b up. *Sum_Frac* module does the fraction part summation. The *Barrel Shift* module stands for the barrel shifter which delivers the output C_{ALM}. These ALM components are illustrated in Fig. 5.2.

From Fig. 5.1(a) we can see, the fraction summation part (*Sum_Frac* module) is the most power dense unit. However, it just takes up about 12.7% of the total power consumption and 9.5% of the area. On the other hand, the barrel shifter unit is not the most power dense component in the ALM structure, but dominates the power consumption of the ALM design, taking up 58.2% of the total power consumption and has higher power density ($0.072\mu W/\mu m^2$) compared to the fixed-point baseline ($0.051\mu W/\mu m^2$). Besides, the *Sum_Frac* module which has the highest power density, should be re-designed and be replaced with modules which have less power density.

Based on the above observation, we propose to split the large barrel shifter into two small barrel shifters. In this way, we can resolve both of these aforementioned issues. Specifically, we rewrite the math expression of the ALM: (4.1) as follows:

$$C_{ALM} = \begin{cases} 2^{k_a+k_b} \cdot (1+x+y) & \\ = 2^{k_a} \cdot (1+x) \cdot 2^{k_b} + 2^{k_b} \cdot y \cdot 2^{k_a} & \\ = A \cdot 2^{k_b} + (B - 2^{k_b}) \cdot 2^{k_a}, & x+y<1, \\ 2^{k_a+k_b+1} \cdot (x+y) & \\ = 2^{k_a} \cdot (1+x) \cdot 2^{k_b+1} + 2^{k_b} \cdot y \cdot 2^{k_a+1} & \\ = (A - 2^{k_a}) \cdot 2^{k_b+1} + (B - 2^{k_b}) \cdot 2^{k_a+1}, & x+y \geq 1 \end{cases} \tag{5.1}$$

As we can see, compared to (4.1), one time $k_a + k_b$ bit shifting operation is separated to a k_a bit shifting and a k_b bit shifting operation.

We show the PAALM design in Fig. 5.3. Unlike the conventional ALM, the LOD blocks in the PAALM structure not only detect the leading one bit and output the exponential and fractional parts, but also deliver the $A - 2^{k_a}$ and $B - 2^{k_b}$ according to (5.1). In this way, the barrel shifting operations are split into two shifting operations. But the real

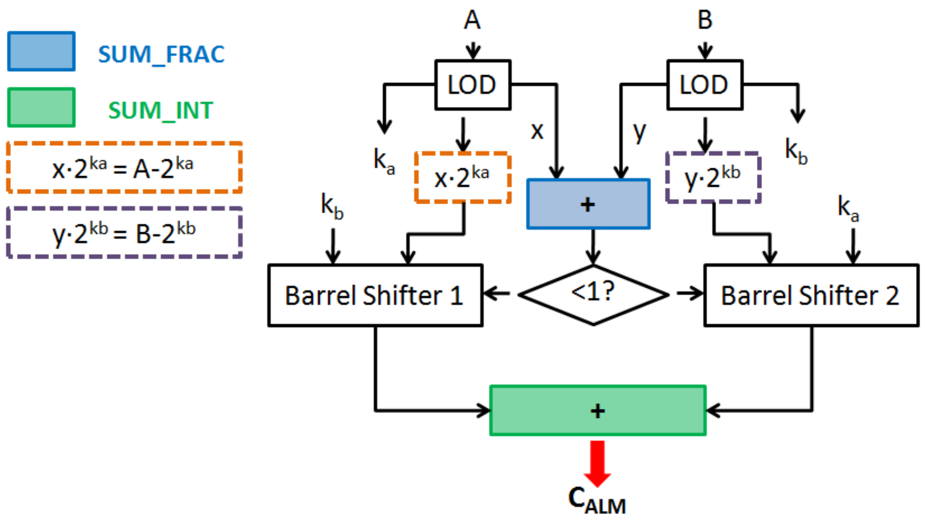


Figure 5.3: The power density aware ALM design

benefits is that the area increase from two barrel shifters will exceed their power increase, which leads to lower power density. After barrel shifting, the PAALM obtains $A \cdot 2^{k_b}$ or $(A - 2^{k_a}) \cdot 2^{k_b}$ depending on whether $x + y$ is smaller or larger than one, using a smaller barrel shifter to obtain two partial products. Then we add these two partial products up to achieve the final output C_{ALM} which is just the same as the output from the conventional ALM. For the most power dense unit *Sum_Frac*, we introduce the inexact summation to replace the exact adder to reduce power density. We will give more details in Sec. 5.2.2.

Fig. 5.1(b) shows the power, area and power density of the components in the new PAALM design. As we can see, the highest power density has been reduced from $0.088\mu W/\mu m^2$ to $0.058\mu W/\mu m^2$. Second, the new *Sum_INT* unit and the improved *SUM_Frac* unit have much lower power density ($0.042\mu W/\mu m^2$) compared to the initial *Sum_Frac* ($0.088\mu W/\mu m^2$) and *Sum_Exp* ($0.047\mu W/\mu m^2$) units. Third, the two small barrel shifters have lower power density than the previous large barrel shifter, which is beneficial to the overall power density. Fourth, the power density differences between different components are much smaller than the previous design, which is good for reducing some local hot spots (although this is less relevant as we assume that the typical thermal analysis granularity will be at the gate level). But the most important factor is that, we end up with reducing the power density of the conventional ALM from $0.069\mu W/\mu m^2$ to $0.051\mu W/\mu m^2$ of PAALM without hurting the accuracy.

5.2.2 The Error Compensation Scheme

To split one large barrel shifter and replace it with two small barrel shifters, the PAALM actually introduces some area overheads, which will be demonstrated in Sec. 5.3. To mitigate this issue, also reduce the power density of the *SUM_Frac* unit which is mentioned in Sec. 2.3.1, we replace the exact summation of $x+y$ with simple inexact summation. Here, we select the simplest truncation adder (TA) to save resource. We use k to represent the number of bits truncated in the $x+y$ summation. As implementing the inexact summation will introduce extra accuracy loss, we need to do error compensation.

However, since the PAALM does the bit shifting operation separately instead of doing just once, we cannot do error compensation, like adding error coefficients directly to the fraction summation part $x+y$, which was usually tried in previous log-based multiplication works like [9, 39]. To do special error compensation for the PAALM design, we replace one LOD unit with a *SO-LOD* (set one LOD) unit, which is shown in Fig. 5.4. Different from the conventional LOD unit, SO-LOD not only generates the exponential part and the fractional part, but also set all the bits of $A-2^{k_a}$ part to bit '1'. As $A-2^{k_a}$ is equivalent to $x\cdot 2^{k_a}$, we rewrite this part as $x'\cdot 2^{k_a}$ and show the structure of PAALM design with error

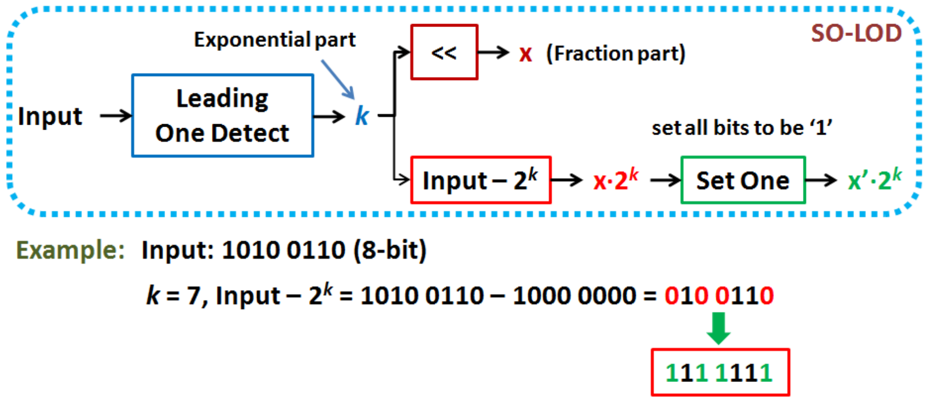


Figure 5.4: The set one LOD (SO-LOD) unit structure

compensation in Fig.5.5. Note that we just replace one of the two LOD units in PAALM with the SO-LOD unit to avoid "over compensation".

5.3 Experimental Results and Discussions

In this section, we evaluate the performance of the proposed power density aware approximate logarithmic multiplication approach, named *PAALM* under 8-bit and 16-bit precision. We also compare the proposed approach against the fixed-point exact multiplier baseline and the conventional ALM design under the same precision.

5.3.1 Experimental Setup

To evaluate the performance of the proposed PAALM design, we demonstrate the hardware and error metrics of PAALM in Table 5.2 and Table 5.3. We also compare PAALM with other state of art improved ALMs. These improved ALMs include: LeAp [9], REALM [39], ILM-EA [3], ILM-AA [3], HEALM-TA, HEALM-SOA.

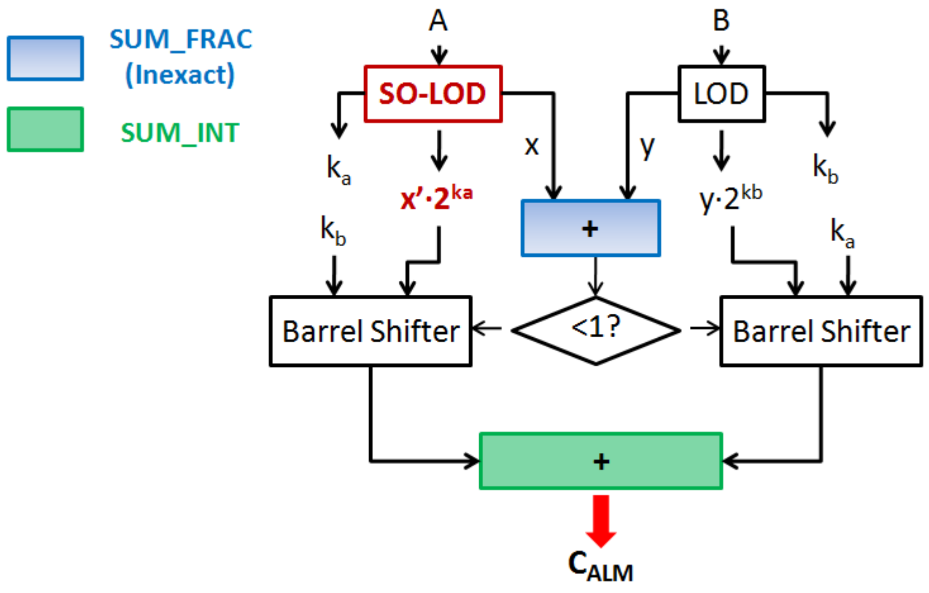


Figure 5.5: The power density aware ALM design with error compensation

All the multipliers listed in Table 5.2 and Table 5.3 are implemented in Verilog HDL and synthesized with Synopsys Design Compiler using EDK 32nm standard cell library [11] as single-cycle designs. As different working frequency will lead to different power consumption of the circuit, and further introduce difficulties to compare the power density. We constrain the throughput (delay) of all the listed multipliers in each table to be the same. Note that different multiplication approaches require different computing latency (Lat) to give outputs, the working frequency ($Freq$) actually will be different for these listed approaches ($Freq = Lat/Delay$). Here, we select the delay/throughput of the fixed-point exact multiplier, which is the baseline, as standard. The critical paths for the 8-bit and 16-bit fixed-point multiplier are 2.07ns (5 cycles) and 4.51ns (7 cycles), respectively; so the delay are 10.35ns ($2.07ns \times 5$) and 31.57ns ($4.51ns \times 7$), respectively. Under these time constraints, we obtain the hardware performance for all the multipliers, including area

Table 5.2: Design metrics for the 8-bit logarithmic multipliers with the same throughput

Approach	***k***	**Lat.**	**Area (μm^2)**	**Power Density ($\mu W/\mu m^2$)**	**Err. Comp.**	**Mean Error /%**	**Peak Error /%**
FxP (baseline)	/	5	1754.10	0.0512	/	/	/
ALM [31]	0	3	820.63	0.0693	w/o	3.76	11.11
PAALM (proposed)	0	2	1391.44	0.0507	w/o	3.76	11.11
	3	2	1326.89	0.0508	w/o	3.77	11.77
	3	2	1339.85	0.0505	w/t	2.96	11.77
	4	2	1274.79	0.0466	w/o	3.83	12.59
	4	2	1287.24	0.0455	w/t	3.27	12.59
	5	2	1212.52	0.0453	w/o	4.09	14.05
	5	2	1245.56	0.0503	w/t	3.52	14.05
	6	2	1199.81	0.0453	w/o	5.13	16.43
	6	2	1201.08	0.0467	w/t	4.42	16.43
Other state of art improved logarithmic multipliers							
LeAp [9]	0	3	1040.21	0.0831	w/t	1.38	4.71
REALM [39]	0	3	1235.14	0.0821	w/t	0.90	3.96
	4	3	709.57	0.0752	w/t	6.58	23.07
ILM-EA [3]	0	3	1221.92	0.0718	w/o	2.84	11.11
ILM-AA [3]	4	3	887.47	0.0708	w/o	5.47	23.47
HEALM-TA	4	3	595.46	0.0702	w/t	3.66	13.77
HEALM-SOA	4	3	664.33	0.0729	w/t	3.12	12.17

and power density (*Power*/*Area*) aspects. As *Power* and *Energy* for all the multipliers can be easily calculated from the products of $Area \times PowerDensity$ and $Power \times Delay$, respectively. To save space, these two metrics will no be listed in the table.

To evaluate the error metrics, we develop behavioral simulation models for all the multipliers listed in Table 5.2 and Table 5.3 in MATLAB and measure the accuracy using 1 million (for 8-bit precision) and 10 million (for 16-bit precision) random inputs uniformly distributed over the set $\{0, 1, ..., (2^8 - 1)\}$ and the set $\{0, 1, ..., (2^{16} - 1)\}$, respectively. The errors are reported with respect to the exact results. The error metrics used to report the error behavior include: mean error (mean of absolute relative error, also referred as MRED in some previous works [3]); and peak error (maximum value of the absolute relative error). All the error metrics are in percentages.

Table 5.3: Design metrics for the 16-bit logarithmic multipliers with the same throughput

Approach	k	Lat.	Area (μm^2)	Power Density ($\mu W/\mu m^2$)	Err. Comp.	Mean Error /%	Peak Error /%
FxP (baseline)	/	7	8753.48	0.0369	/	/	/
ALM [31]	0	3	1714.20	0.0407	w/o	3.85	11.11
PAALM (proposed)	0	2	3296.25	0.0360	w/o	3.85	11.11
	9	2	2939.18	0.0369	w/o	3.85	11.34
	9	2	2978.31	0.0365	w/t	3.43	11.34
	10	2	2836.50	0.0363	w/o	3.85	11.56
	10	2	2957.98	0.0361	w/t	3.15	11.56
	11	2	2733.06	0.0368	w/o	3.88	12.00
	11	2	2898.26	0.0368	w/t	2.92	12.00
	12	2	2678.93	0.0365	w/o	3.96	12.82
	12	2	2798.89	0.0361	w/t	3.43	12.82
	13	2	2642.34	0.0363	w/o	4.29	14.28
	13	2	2743.48	0.0357	w/t	3.70	14.28
	14	2	2553.89	0.0348	w/o	5.48	16.67
	14	2	2619.46	0.0359	w/t	4.62	16.67
Other state of art improved logarithmic multipliers							
LeAp [9]	0	3	1990.71	0.0455	w/t	0.98	4.76
REALM [39]	0	3	2383.36	0.0488	w/t	0.75	3.70
	9	3	1572.90	0.0423	w/t	1.06	5.27
HEALM-TA	9	2	1511.39	0.0410	w/t	1.64	5.83
HEALM-SOA	9	2	1577.47	0.0412	w/t	1.38	5.15

5.3.2 Performance Evaluation and Comparison

We show the hardware performance of the proposed PAALM design under 8-bit precision in Table 5.2. Note that as PAALM will introduce area overheads when compared to the conventional ALM design. We introduce inexact summation and error compensation scheme to save resource and improve error metrics as mentioned in Sec. 5.2.2. The parameter k indicates the number of bits truncated in the $x+y$ summation. In the *Err. Comp.* column: the "w/t", "w/o" mark represent "with" or "without" error compensation, respectively. By introducing inexact summation, from Table 5.2, we observe that when compared to the exact multiplier baseline, PAALM can improve the power density and area up to 11.5% and 31.6%, respectively. Also when compared to the conventional ALM design, PAALM will improve the power density up to 34.6% with area overheads and outperforms all the listed state of art works in power density.

For the 16-bit multipliers, the hardware performance and error metrics are demonstrated in Table 5.3. PAALM reduces 5.7% power density and 70.8% area when compared to the exact multiplier baseline; also reduces up to 14.5% of power density when compared with the ALM design with area overheads. Similar to the 8-bit case, 16-bit PAALM also outperforms all the listed state of art works in respect of the power density.

The error metrics of 8-bit and 16-bit PAALM design are also demonstrated in Table 5.2 and Table 5.3. As proved in Sec. 2.3.1, without error compensation, PAALM shows completely the same error behavior as the conventional ALM. However, different from previous works, the error compensation scheme (mentioned in Sec. 5.2.2) for PAALM will do little improvement to the mean and peak error, but focus on improving the error bias. We show the curves for the absolute value of the error bias of PAALM with error compensation under 8-bit and 16-bit precision with different k values in Fig. 5.6. By observing Fig. 5.6, we can find an obvious convex property of the curves, not monotonically decreasing when we decrease the k value. The reason is that when k is small, the "set one" error compensation scheme will introduce larger and larger positive error to balance the negative error of ALM when we increase k. However, when the error bias reaches the minimum value: -0.17 (when $k = 4$ with 8-bit precision) or 0.08 (when $k = 12$ with 16-bit precision), and we further increase k, the absolute value of the positive error will become larger than which of the negative error, thus leads to a "rebound" of the error bias curve.

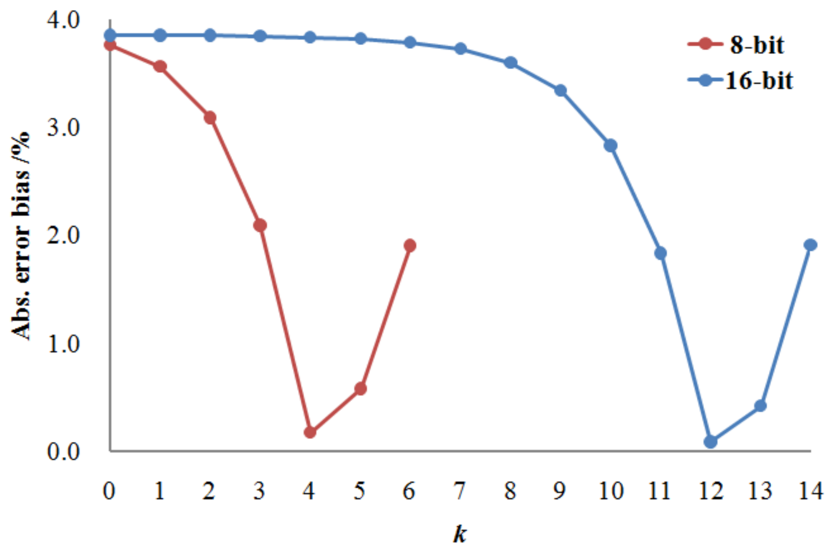


Figure 5.6: The absolute error bias values of 8-bit and 16-bit error compensated PAALM for different k values

5.3.3 Evaluation on CNN Application

To further evaluate the performance of PAALM in applications which are multiplication-intensive, we first implement the multiplier into a convolutional neural network (CNN) and compare with the fixed-point multiplier baseline and the conventional ALM. The application is developed on a Python platform.

We use CIFAR10 [23] dataset to test the CNN application with approximate multipliers. The network consists of two convolution (CONV) layers and three fully connected (FC) layers. The network is trained with 3000 steps, using double-precision floating point numbers with a batch size of 128 in one step. To test the performance of the approximate multipliers, we replace the floating-point multipliers with PAALM, ALM and fixed-point multipliers in 8-bit precision to do the inference. Since the CONV layer is computation-intensive and FC layer is memory-extensive, we simply replace the multipliers in CONV

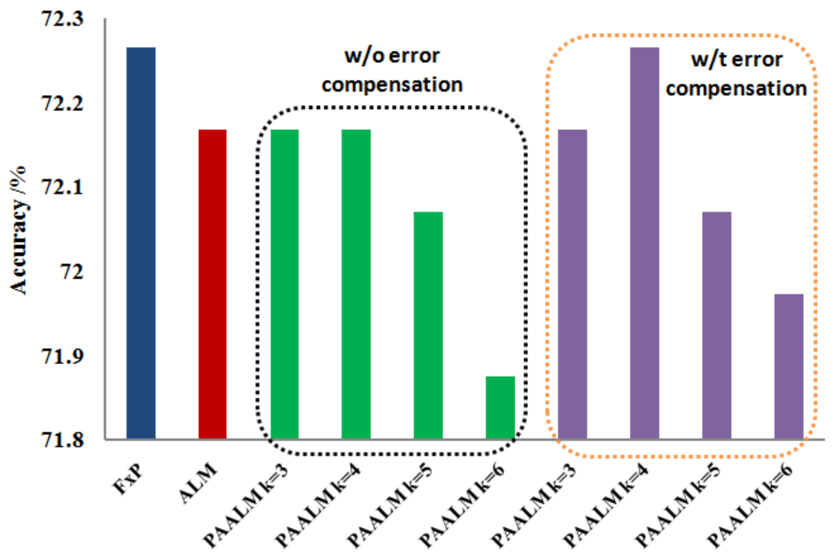


Figure 5.7: CIFAR10 dataset inference accuracy based on different approximate multipliers

layers. Fig. 5.7 shows the classification accuracy comparison of PAALM, conventional ALM and fixed-point multiplier on 1024 images. The accuracy is in percentage.

From Fig. 5.7, we can observe that all the listed multipliers can achieve similar inference accuracy. Without error compensation, PAALM will lose some accuracy when inexact summation is introduced; while when $k < 5$, PAALM can perform as well as the conventional ALM. If error compensation is considered, we notice that when $k = 4$, PAALM can outperform the conventional ALM, and achieves the same inference accuracy as the fixed-point multiplier baseline. This is due to the extremely low error bias (almost zero) as demonstrated in Fig. 5.6.

5.3.4 Evaluation on Image Processing Application

Now, we show how the proposed *PAALM* design compare to the conventional ALM and fixed-point multiplier baseline in a multimedia application. Discrete cosine transformation (DCT) is a commonly used lossy image compression method. The quality of the compressed images is usually evaluated using metrics such as PSNR (peak signal noise ratio) and higher PSNR value represents better image quality. We implement the proposed PAALM design with 8-bit precision in the DCT-iDCT (inverse DCT) workloads, and compare with the conventional ALM and fixed-point multiplier on five example images. As the error bias behavior of error compensated PAALM with different k value shows a convex trend, and performs the best when k equals to 4. We just list the experimental results from $k = 3 \sim 6$.

We show results of image compression in Table 5.4. The results show that without error compensation, the image quality will worsen when the value of k increases. While if error compensation is introduced, when k equals to 4, thanks to a more balanced error behavior, we can achieve the best image quality of 27.4dB in average, improving 8.6dB when compared to the conventional ALM design. Note that the error compensation for PAALM introduces almost no area and power overheads, thus we obtain an approximate multiplier which achieves a balance of resource efficiency and low power density.

Table 5.4: PSNR (dB) for images after DCT-iDCT using logarithmic multipliers

Approach	***k***	**Err. Comp.**	**Lena**	**Boat**	**Barbara**	**House**	**Pepper**	**Avg.**
FxP (baseline)	/	/	40.3	39.7	39.6	40.1	38.5	39.6
ALM	0	w/o	19.1	18.7	19.3	18.4	18.7	18.8
PAALM (proposed)	0	w/o	19.1	18.7	19.3	18.4	18.7	18.8
	3	w/o	19.0	18.6	19.3	18.4	18.6	18.8
	3	w/t	21.7	21.2	21.9	21.3	21.5	21.5
	4	w/o	19.1	18.7	19.3	18.4	18.7	18.8
	4	**w/t**	**27.5**	**26.5**	**27.0**	**28.8**	**27.0**	**27.4**
	5	w/o	18.4	18.1	18.6	17.1	17.9	18.0
	5	w/t	24.8	23.5	24.5	27.4	24.7	25.0
	6	w/o	16.3	16.1	16.4	14.9	15.9	15.9
	6	w/t	20.5	20.0	20.2	17.4	19.2	19.5

5.4 Summary

The work in this chapter proposes a novel power density aware approximate logarithmic multiplier (*PAALM*) design to mitigate the high power density issue occurred in the existing ALM designs. By re-designing the high computing switch activities of existing ALM designs based on equivalent mathematical formula, the power density can be reduced with no accuracy loss but with some area overheads. Experimental results shows that the proposed PAALM design can improve the power density and the area at the same time when compared with the fixed-point multiplier baseline. The PAALM design can also achieve extremely low error bias with error compensation. Furthermore, the PAALM shows good performance when implemented in CNN and image processing applications.

Chapter 6

Conclusions

Approximate computing enables efficient trade-off among accuracy, area, latency and power for error tolerant applications implementation. However, most of the existing works lack the systematic configurability for accuracy vs. area/power/latency trade-off. This thesis introduced two related studies focusing on extension of counter-based stochastic computing, and two studies focusing on improvement of the conventional logarithmic multiplier, which is helpful and meaningful to more efficient approximate arithmetic circuit designs. The contributions of each study is summarized in this chapter.

6.1 Counter-Based Stochastic-Behaving Approximate Integer Multiplier Design and Scaling Technique

This work proposes a new counter-based stochastic-behaving approximate integer unsigned multiplier, *COSAIM* for many emerging machine learning hardware implementation. *COSAIM* is an accelerated design of recently proposed counter-based stochastic multiplier. As a result, it still keeps the advantages of stochastic computing such as built-in progress reconfigurability for progressive performance-accuracy trade-off. The work in this chapter also proposes a novel scaled counting-based stochastic computing multiplication design, named *Scaled-CBSC*. The proposed design introduces scaling bits to promise the inputs of the SC multiplication kernel to be larger than 0.5, avoiding the “small” number region which leads to large relative error of SC multiplication. Numerical results shows that 8-bit *Scaled-CBSC* and Scaled COSAIM designs with 3 scaling bits can achieve up to 46.6% and 30.4% improvements in mean error and standard deviation, respectively; reduce the peak relative error from 100% to 1.8%. For hardware performance, the *Scaled-CBSC* can improve 12.6%, 51.5%, 57.6%, 58.4% in delay, area, area-delay product, energy consumption, respectively, over the state of art work. Though COSAIM will introduce some resource overhead, it can significantly reduce the delay and the area-delay product. For discrete cosine transformation (DCT) application, 3 scaling bits are required for 8-bit *Scaled-CBSC* and COSAIM multiplication to significantly improves the image quality by 5.9dB.

6.2 A Counting-Based Stochastic Computing Division

This work proposes a fast and energy efficient approximate divider design based on stochastic computing division design, CBDIV, which exploits both the correlation requirement of existing SC-based division methods and the high efficiency of counting-based SC scheme. CBDIV fits well with the hybrid computing in which binary and SC implementations are both required for overall better application performance, like image processing. Experimental results show that the proposed CBDIV outperforms state of art works by

77.8% in accuracy, 37.1% in delay, 21.5% in area, 50.6% in ADP and 25.9% in power. CBDIV also saves 31.9% in energy consumption when compared to the fixed-point division baseline, and is much more energy efficient than early proposed SC-based dividers if the inputs and outputs are both binary numbers. Furthermore, CBDIV with 5-bit precision can even outperform state of art works with 7-bit precision in accuracy by 15.4%. Finally, we compare CBDIV with other state of art SC dividers in contrast stretch application and shows that CBDIV can improve the accuracy with 20.6dB in average, which is a huge improvement.

6.3 Hardware Efficient Approximate Logarithmic Multiplier With Reduced Error

This work proposes a novel hardware-efficient approximate logarithmic multiplier, called *HEALM*. The proposed design, first determines the truncation width for mantissa summation in ALM. Then the error reduction is performed via a lookup table for multiple partitioned input ranges. Numerical results shows that *HEALM* and its enhanced designs can lead to more accurate results with reduced area and power at the same time than the existing ALM baseline design. It also outperforms the state of art design, REALM, with up to 2.92%, 9.30%, 16.08%, 17.61% improvement in mean error, peak error, area, power consumption for 8-bit precision. For discrete cosine transformation (DCT) application, with different values of k, HEALM-TA can improve the image quality upon the ALM baseline by 7.8$\sim$17.2dB in average and HEALM-SOA could improve 2.9$\sim$15.8dB in average, respectively. Besides, HEALM-TA and HEALM-SOA outperforms all the state of art works with $k = 2, 3, 4$ on the image quality.

6.4 Power Density Aware Approximate Logarithmic Multiplier Design

This work proposes a novel power density aware approximate logarithmic multiplier (*PAALM*) design to mitigate the high power density issue occurred in the existing ALM designs. By re-designing the high computing switch activities of existing ALM designs based on equivalent mathematical formula, the power density can be reduced with no accuracy loss but with some area overheads. Experimental results shows that the proposed PAALM design can improve the power density and the area at the same time when compared with the fixed-point multiplier baseline. The PAALM design can also achieve extremely low error bias with error compensation. Furthermore, the PAALM shows good performance when implemented in CNN and image processing applications.

Bibliography

[1] Armin Alaghi and John P Hayes. Survey of stochastic computing. *ACM Transactions on Embedded computing systems (TECS)*, 12(2s):1–19, 2013.

[2] Armin Alaghi and John P Hayes. Fast and accurate computation using stochastic circuits. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–4. IEEE, 2014.

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

[4] Kartikeya Bhardwaj, Pravin S Mane, and Jörg Henkel. Power-and area-efficient approximate wallace tree multiplier for error-resilient systems. In *Fifteenth International Symposium on Quality Electronic Design*, pages 263–269. IEEE, 2014.

[5] Te-Hsuan Chen and John P Hayes. Design of division circuits for stochastic computing. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 116–121. IEEE, 2016.

[6] Zhiyuan Chen, Yufei Ma, and Zhongfeng Wang. Optimizing stochastic computing for low latency inference of convolutional neural networks. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–7, 2020.

[7] Chen, Yu-Hsin and Krishna, Tushar and Emer, Joel and Sze, Vivienne. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. In *IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers*, pages 262–263, 2016.

[8] Shao-I Chu, Yi-Ming Lee, Chen-En Hsieh, Jiun-Han Yen, and Yu-Jung Huang. Stochastic circuit design of image contrast stretching. In *2019 International SoC Design Conference (ISOCC)*, pages 81–82. IEEE, 2019.

[9] Zahra Ebrahimi, Salim Ullah, and Akash Kumar. Leap: Leading-one detection-based softcore approximate multipliers with tunable accuracy. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 605–610. IEEE, 2020.

[10] Brian R Gaines. Stochastic computing systems. In *Advances in information systems science*, pages 37–172. Springer, 1969.

[11] R Goldman, K Bartleson, T Wood, K Kranen, V Melikyan, and E Babayan. 32/28nm educational design kit: Capabilities, deployment and future. In *2013 IEEE Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia)*, pages 284–288. IEEE, 2013.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[13] Google. Tensor models. *GitHub,[Online]. Available: https://github.com/tensorflow/models.[Accessed 25 02 2018]*, 2016.

[14] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pages 1737–1746, 2015.

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

[16] Guixia He, Jiaquan Gao, and Jun Wang. Efficient dense matrix-vector multiplication on gpu. *Concurrency and Computation: Practice and Experience*, 30(19):e4705, 2018. e4705 CPE-17-0584.R2.

[17] Reza Hojabr, Kamyar Givaki, SM Tayaranian, Parsa Esfahanian, Ahmad Khonsari, Dara Rahmati, and M Hassan Najafi. Skippynn: An embedded stochastic-computing accelerator for convolutional neural networks. In *Proceedings of the 56th Annual Design Automation Conference 2019*, page 132. ACM, 2019.

[18] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[19] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[20] Kyounghoon Kim, Jungki Kim, Joonsang Yu, Jungwoo Seo, Jongeun Lee, and Kiyoung Choi. Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2016.

[21] Kyounghoon Kim, Jongeun Lee, and Kiyoung Choi. An energy-efficient random number generator for stochastic circuits. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 256–261. IEEE, 2016.

[22] Israel Koren. *Computer arithmetic algorithms*. AK Peters/CRC Press, 2018.

[23] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[24] Parag Kulkarni, Puneet Gupta, and Milos Ercegovac. Trading accuracy for power with an underdesigned multiplier architecture. In *2011 24th Internatioal Conference on VLSI Design*, pages 346–351. IEEE, 2011.

[25] Fengfu Li, Bo Zhang, and Liu Bin. Ternary weight networks. *arXiv e-prints*, Nov 2016.

[26] Peng Li, David J Lilja, Weikang Qian, Kia Bazargan, and Marc D Riedel. Computation on stochastic bit streams digital image processing case studies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(3):449–462, 2013.

[27] Zhouhan Lin, Matthieu Courbariaux, and Bengio Yoshua. Neural networks with few multiplications. *arXiv e-prints*, Feb 2016.

[28] Siting Liu and Jie Han. Energy efficient stochastic computing with sobol sequences. In *Proceedings of the Conference on Design, Automation & Test in Europe*, pages 650–653. European Design and Automation Association, 2017.

[29] Siting Liu and Jie Han. Toward energy-efficient stochastic circuits using parallel sobol sequences. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(7):1326–1339, 2018.

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

[31] John N Mitchell. Computer multiplication and division using binary logarithms. *IRE Transactions on Electronic Computers*, (4):512–517, 1962.

[32] Ali Naderi, Shie Mannor, Mohamad Sawan, and Warren J Gross. Delayed stochastic decoding of ldpc codes. *IEEE Transactions on Signal Processing*, 59(11):5617–5626, 2011.

[33] M Hassan Najafi, David J Lilja, and Marc Riedel. Deterministic methods for stochastic computing using low-discrepancy sequences. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2018.

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

[35] Florian Neugebauer, Ilia Polian, and John P Hayes. Building a better random number generator for stochastic computing. In *2017 Euromicro Conference on Digital System Design (DSD)*, pages 1–8. IEEE, 2017.

[36] Bharath Srinivas Prabakaran, Semeen Rehman, Muhammad Abdullah Hanif, Salim Ullah, Ghazal Mazaheri, Akash Kumar, and Muhammad Shafique. Demas: An efficient design methodology for building approximate adders for fpga-based systems. In *2018*

Design, Automation & Test in Europe Conference & Exhibition (DATE), pages 917–920. IEEE, 2018.

[37] Gintaras V Puskorius, Lee Feldkamp, and Leighton Davis. Dynamic neural network methods applied to on-vehicle idle speed control. *Proceedings of the IEEE*, 84(10):1407–1420, 1996.

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

[39] Hassaan Saadat, Haris Javaid, Aleksandar Ignjatovic, and Sri Parameswaran. Realm: reduced-error approximate log-based integer multiplier. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1366–1371. IEEE, 2020.

[40] Hyeonuk Sim and Jongeun Lee. A new stochastic computing multiplier with application to deep convolutional neural networks. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2017.

[41] Hyeonuk Sim, Dong Nguyen, Jongeun Lee, and Kiyoung Choi. Scalable stochastic-computing accelerator for convolutional neural networks. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 696–701. IEEE, 2017.

[42] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[43] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. Efficient processing of deep neural networks. *Synthesis Lectures on Computer Architecture*, 15(2):1–341, 2020.

[44] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, June 2014.

[45] Nikos Temenos and Paul P Sotiriadis. Deterministic finite state machines for stochastic division in unipolar format. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2020.

[46] Salim Ullah, Sanjeev Sripadraj Murthy, and Akash Kumar. Smapproxlib: library of fpga-based approximate multipliers. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[47] Salim Ullah, Semeen Rehman, Bharath Srinivas Prabakaran, Florian Kriebel, Muhammad Abdullah Hanif, Muhammad Shafique, and Akash Kumar. Area-optimized low-latency approximate multipliers for fpga-based hardware accelerators. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[48] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. Improving the speed of neural networks on cpus. 2011.

[49] Swagath Venkataramani, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. Approximate computing and the quest for computing efficiency. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2015.

[50] Junsong Wang, Qiuwen Lou, Xiaofan Zhang, Chao Zhu, Yonghua Lin, and Deming Chen. Design flow of accelerating hybrid extremely low bit-width neural network in embedded fpga. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 163–1636. IEEE, 2018.

[51] Di Wu and Joshua San Miguel. In-stream stochastic division and square root via correlation. In *Proceedings of the 56th Annual Design Automation Conference 2019*, page 162. ACM, 2019.

[52] Georgios Zervakis, Iraklis Anagnostopoulos, Sami Alsalamin, Ourania Spantidi, Isai Roman-Ballesteros, Jorg Henkel, and Hussam Amrouch. Thermal-aware design for approximate dnn accelerators. *IEEE Transactions on Computers*, pages 1–1, 2022.

[53] H. Zhou, S. P. Khatri, J. Hu, and F. Liu. Scaled population arithmetic for efficient stochastic computing. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 611–616, 2020.