Hardware-Efficient Logarithmic Floating-Point Multipliers for Error-Tolerant Applications

by

Zijing Niu

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Integrated Circuits and Systems

Department of Electrical and Computer Engineering
University of Alberta

Abstract

The deployment of computation-intensive applications, such as digital signal processing (DSP) and machine-learning (ML), on resource-constrained applications is spurring research that aims to increase the power efficiency of the arithmetic circuits that perform a huge amount of computation. Approximate computing is one of the emerging strategies to reach this objective. Neural networks (NNs) involve extensive multiply-and-accumulate computations, especially in the training process, where iterative computations incur significant energy consumption. DSP applications, such as image processing, also demands intensive arithmetic computations. In this research project, we focus on the design of hardware-efficient floating-point (FP) multipliers for the energy-efficient implementation of error-tolerant computation-intensive applications using approximate computing techniques.

As an alternative to the conventional FP representation, the logarithmic representation of FP numbers has been considered for the acceleration of NNs. We show that the FP representation is naturally suited for the binary logarithm of numbers and, thus, logarithmic arithmetic. In this thesis, two logarithmic approximation methods are proposed to generate double-sided error distributions that mitigate the accumulative effect of errors introduced in both the logarithm conversion and the anti-logarithm conversion. Hardware-efficient logarithmic FP multipliers are then proposed by using simple operators, such as adders and multiplexers, to replace complex conventional FP multipliers. The radix-4 logarithm is considered to further reduce the hardware complexity. The proposed multipliers provide superior trade-offs between accuracy and hardware, with up to 30.8% higher accuracy than a recent logarithmic FP design

or up to 68× less energy than the conventional FP multiplier. Using the proposed FP logarithmic multipliers in JPEG image compression achieves higher image quality than the recent multiplier design with up to 4.7 dB larger peak signal noise ratio. For training in benchmark NN applications, including a 922-neuron model for the MNIST dataset, the proposed FP multipliers can slightly improve the classification accuracy while achieving 4.2× less energy and 2.2× smaller area than a state-of-the-art design.

Preface

This dissertation presents original work in the field of approximate computing by Zijing Niu.

The review of approximate multipliers in Chapter 2 was partially published as T. Zhang, Z. Niu and J. Han, "A Brief Review of Logarithmic Multiplier Designs," IEEE 23rd Latin American Test Symposium (LATS), 2022, pp. 1-4. T. Zhang and I collaboratively reviewed literature and drafted the article.

One of the logarithmic multiplier designs, presented in Chapter 3 to Chapter 6, was published in GLSVLSI'21, Proceedings of the 31st IEEE/ACM Great Lakes Symposium on VLSI, 2021, as "A Logarithmic Floating-Point Multiplier for the Efficient Training of Neural Networks." Z. Niu, H. Jiang, M. Ansari, B. Cockburn, L. Liu, and J. Han. I devised the logarithmic multiplier, carried out the simulations and circuit synthesis, and composed the article. I also presented the article at the conference. Dr. H. Jiang provided an original idea of improving the logarithmic multiplier designs and revised the manuscript. Dr. J. Han supervised this work and revised the manuscript together with Dr. B. Cockburn, Dr. M. Ansari and Dr. L. Liu. The other logarithmic multiplier designs, presented in Chapter 3 to Chapter 6, are presented in a manuscript in preparation for submission. I devised the logarithmic multiplier designs, performed the error analysis and circuit synthesis, implemented error-tolerant applications, including neural networks and JPEG compression, using the proposed designs and completed the manuscript. T. Zhang provided many technical suggestions for improving the design and manuscript. H. Jiang provided the MATLAB code for JPEG compression and useful suggestions for the simulation of

neural network applications. Dr. J. Han and Dr. B. Cockburn provided valuable suggestions to improve the structure and the technical content of the manuscript.

Acknowledgements

Words cannot express my deepest gratitude to my supervisor Dr. Jie Han for his generous guidance and unlimited support throughout my graduate study. Thanks for providing insights and professional advice for solving research problems, offering invaluable advice to my personal development, and always being there when I needed support. I could not have undertaken this journey without Dr. Jie Han.

I greatly appreciate my collaborator Dr. Bruce Cockburn for his valuable suggestions on my research and manuscript writing. Thanks for his words of encouragement; it means a lot to me. I would also like to thank Dr. Honglan Jiang for her generous support and feedback on my publications. I learned so much from her work and the discussions with her.

I am grateful to all my colleagues in the research group and friends for their friendship and help. Many thanks to Tingting Zhang, Chengcheng Tang, Bailiang Liu, and Qichao Tao. Special thanks to Tingting Zhang, for her patience, late-night feedback sessions and moral support during my graduate study. Thanks to Zeyu Sun, Xiang Li, and Yuxiang Wang. Our friendships have added a lot to my life.

Most importantly, thanks to my beloved family for their unconditional love, understanding and financial support. My greatest thanks to my partner, Zhitao Nie, for his consistent love, company and encouragement. Their belief in me has kept my spirits and motivation high during this journey.

Lastly, I would like to thank my lovely cat, bubu, for all the entertainment and emotional support.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AC Approximate Computing.

AE Average Error.

AMBE Approximate Modified Booth Encoding in [1].

Bfloat16 Brain FP.

CFPU Configurable Floating-point Unit in [2].

CMOS Metal–oxide Semiconductor.

DCT Discrete Cosine Transform.

DSP Digital Signal Processing.

FP Floating-Point.

FPM Floating-point Multiplier.

IDCT Inverse Discrete Cosine Transform.

ILM Iterative Logarithmic Multiplier.

JPEG Joint Photographic Experts Group.

KNN K-nearest Neighbor.

LAM Logarithmic Approximate Floating-point Multiplier in [3].

LM Logarithmic Multiplier.

LOA Lower-part-Or Adder in [4].

MAA3 Approximate Mirror Adder in [5].

MAC Multiply-ACcumulate.

ML Machine Learning.

MLP Multi-layer Perceptron.

MRED Mean Relative Error Distance.

NN Neural Network.

NOD Nearest-one Detector in [6].

PSNR Peak Signal Noise Ratio.

R4L Radix-4 Logarithm.

ReLU Rectified Linear Unit.

Chapter 1

Introduction

1.1 Motivation

Due to further scaling of complementary metal–oxide semiconductor (CMOS) technology, power density has significantly increased in integrated circuits. The substantial increase in energy becomes a limiting factor for computation-intensive applications, such as machine learning (ML) and digital signal processing (DSP), in resource-constrained devices [7][8]. The computations in many important recent applications involves massive multiply-accumulate (MAC) operations, which requires a significant amount of energy. This challenge motivates the development of more efficient arithmetic circuit implementations for emerging computing systems.

Many computation-intensive applications, such as image processing and neural networks (NNs), that produce results for human perception, can successfully tolerate a degree of computational error [9]. Approximate computing (AC) has emerged as a promising strategy to improve the energy efficiency in such systems [10]. In particular, AC has extensively been exploited in hardware implementations, including approximate arithmetic circuit designs, voltage over-scaling and precision scaling techniques [11], [12].

Floating-point (FP) arithmetic is often favored in applications that require sufficient relative accuracy over a wide dynamic range. Moreover, since FP MAC circuits, especially the multipliers therein, dominate the power dissipation and circuit area,

efficient FP multiplier design has been of interest [1]–[3], [13]–[15]. Nevertheless, FP multipliers are relatively underexplored compared to their fixed-point counterparts and only a few of them have been applied to applications, such as image processing and, particularly, the training process of NNs. Bit width reduction of the FP representation for the training of NNs increase efficiency [16][17]. However, it can cause a significant deterioration on accuracy. Therefore, designs of approximate FP multipliers are promising to further improve the hardware performance for error-tolerant applications.

As an alternative to the conventional FP representation, logarithmic representations of FP numbers have been considered for the acceleration of NNs. For example, Lognet shows that logarithmic computation can enable more accurate encoding of weights and activations that results in higher classification accuracies at low resolutions [18]. A state-of-the-art 4-bit training strategy for deep NNs was developed for the logarithmic radix-4 representation [19]. Hence, efficient FP logarithmic multipliers (LMs) have become promising for NN training. Significant progress has been made in exploiting reduced-precision integers for inference, while 8-bit FP numbers [20] and logarithmic 4-bit FP numbers [19] have shown their effectiveness in the training of deep NNs.

1.2 Thesis Contributions

In this research project, we propose five energy-efficient FP LMs for error-tolerant, computation-intensive applications. Two novel approximation methods are described for the logarithm and anti-logarithm conversions that generate double-sided error distributions. The multiplier designs consist of simple operators, such as adders and multiplexers, which lead to lower power consumption and area cost compared to the conventional FP multipliers. Finally, the application-level performance of the proposed FP multipliers is evaluated for an image processing application and several benchmark NNs. It was also found that the classification accuracy can be slightly

improved in some cases with a significant reduction in energy consumption.

The novel contributions in this work are summarized as follows:

- Two novel approximation methods that generate double-sided error distributions in logarithm and anti-logarithm conversions to minimize the accumulation error.
- Five FP LMs are designed with circuit optimizations. Especially, a radix-4 logarithm (R4L) is developed for further reducing the hardware complexity.
- A detailed and comprehensive error evaluation is presented for four different FP precision formats. In particular, the relation between the FP precision format and the error behavior of the multipliers is analyzed.
- The proposed designs are evaluated in the JPEG compression and the training process of NNs to improve the hardware efficiency.

1.3 Thesis Outline

The remainder of this thesis is organized as follows. In Chapter 2, the background and the related work are reviewed. Chapter 3 introduces the proposed logarithmic approximation methods and logarithmic multiplications. A detailed error analysis for the approximation error and the proposed logarithmic multiplications are also presented. Circuit designs of five proposed FP multipliers and the radix-4 logarithm hardware reduction approach are illustrated in Chapter 4. Their performance in accuracy and hardware are also shown. Chapter 5 presents case studies for error-tolerant applications. Finally, conclusions are provided in Chapter 6.

Chapter 2

Background

In this chapter, the theory that underlies logarithm approximation, various approximate multipliers and the background on the error-tolerant applications are introduced.

2.1 Theory of Logarithm Approximation

Let Z be a number in the binary representation $Z = 2^e(1 + k)$, where k $(0 \leq k < 1)$ denotes the fractional part and e is the exponent. Mitchell first proposed a simple method for a logarithmic conversion, given by [21]:

$$\log_2 Z \cong e + k. \tag{2.1}$$

If $\log_2 (1 + k)$ is approximated by l, then Mitchell's anti-logarithm approximation can be expressed as:

$$2^l \cong l + 1, \tag{2.2}$$

where $0 \leq l < 1$.

Mitchell's approximation offers both high speed and low hardware cost. However, it always underestimates the true logarithm, which cause negative errors to accumulate in MAC operations. To produce a double-sided error distribution, a nearest-one logarithmic approximation finds the nearest power of two for Z [6]. When $Z - 2^e < 2^{e+1} - Z$, it uses the same logarithm as Mitchell's method; otherwise, Z is given by $Z = 2^{e+1}(1 - y)$, where $0 \leq y < 0.25$, and thus the logarithm is approximated as:

$$\log_2 Z \cong e + 1 - y. \tag{2.3}$$

2.2 Review of Approximate Multipliers

2.2.1 Approximate Logarithmic Multipliers (LMs)

Mitchell's approximation has been the basis for many proposed LMs. Operand decomposition improves the accuracy of Mitchell's approximation by dividing the two inputs into four inputs to reduce the number of '1's in the inputs [22]. An improved operand decomposition algorithm further reduces the energy consumption in LMs [23]. The iterative LM (ILM) improves the accuracy through a pipelined implementation with an error correction circuit that leads to iterative calculations of compensation terms [24]. A truncated ILM further reduces the hardware complexity [25]. A low-cost two-stage ILM compensates for errors in the addition of approximate logarithms [26].

To further improve the hardware efficiency, Mitchell's algorithm has been combined with other approximation techniques and optimized implementations of components. A customizable signed LM utilizes logarithmic approximation and truncation of operands, in which the one's complement representation is adopted to imprecisely handle negative numbers [27]. A cost-efficient two-stage logarithmic design uses a truncated LM in an iterative structure [26]. In both the non-iterative and iterative LMs, approximate adders are used to add the mantissas to reduce energy and improve accuracy [28]. A set-one adder (SOA) sets the lower significant bits as '1's to compensate the accumulated errors in sums generated in Mitchell's LM. The lower significant sum bits are computed using OR gates in the lower-part-or adder (LOA) and are set as one of the inputs in the approximate mirror adder (MAA3) [4] [5]. A two-stage LM employs different trimming strategies in the least significant parts of the input operands and the mantissas of the trimmed operands, which leads to lower energy and area cost [29]. A dynamic range LM relies on a truncation scheme to dynamically

compensate for the accumulated errors generated by Mitchell's approximation [30].

Unlike Mitchell's approximation-based LMs, an improved LM uses the nearest-one logarithmic approximation method to produce double-sided error distributions [6]. A nearest-one detector (NOD) was proposed to first detect the nearest power of two and then produce the approximate logarithm.

2.2.2 Approximate Floating-Point Multipliers

Truncation and voltage over-scaling techniques are commonly used for approximate FP multipliers [31]–[35]. A configurable approximate FP multiplier utilizes the K-nearest neighbor (kNN) algorithm to determine the truncation bits of a given input that can minimize energy and area [13]. Using Mitchell's approximation, a logarithmic approximate FP multiplier (LAM) improves the energy efficiency of NN training [3]. Moreover, configurable FP multipliers have been studied to provide different levels of accuracy in real-time [2] [14]. In [2], a configurable floating-point unit (CFPU) avoids multiplication by discarding one mantissa or adding and shifting two mantissas. The multiplication is also replaced with addition in a runtime configurable FP multiplier, using an accuracy tuning method [14]. However, neither of these two configurable FP multipliers completely eliminates multiplication since exact multiplication is still required when the error rate exceeds a pre-determined value.

Approximate fixed-point multipliers are required for the mantissa multiplication. For example, an approximate modified Booth encoding (AMBE) algorithm generates the partial products and then uses an inexact 4-2 compressor to optimize the mantissa multiplier [1]. This design also adopts bit truncation in the partial products to generate variable accuracy. Three approximate FP units are developed by proposing an approximate speculative multiplier and using gate-level pruning with an approximate speculative adder [15].

2.3 Error-Tolerant Applications

This section presents a brief background for the error-tolerant applications that involve intensive computations and discusses the usage of the hardware-efficient floating-point multipliers in these applications.

2.3.1 JPEG Compression

Joint Photographic Experts Group (JPEG) is a worldwide standard for the compression of digital images. The JPEG lossy compression algorithm uses the discrete cosine transform (DCT) for the encoding and decoding processes [36]. The DCT operation converts each image pixel from the spatial domain into the frequency domain. The high-frequency information is then discarded by using a quantization matrix. The image quality can be user-defined by setting a compression quality factor, which ranges from 1 to 100.

In the JPEG compression, an input image is first partitioned into blocks of 8×8 pixels for the DCT coding. Let the pixel of the input image be $i[x, y]$, an 8×8 DCT coefficient matrix, denoted by $\boldsymbol{I[p, q]}$, for each 8×8 pixel block is obtained by [37]:

$$\boldsymbol{I[p, q]} = \frac{C[p]C[q]}{4} \sum_{x=0}^{7} \sum_{y=0}^{7} i[x, y] \cos \frac{(2x+1)p\pi}{16} \cos \frac{(2y+1)q\pi}{16} \tag{2.4}$$

where $0 \leq p, q \leq 7$, and $C[p]$ (or $C[q]$) is expressed as:

$$C[p] = \begin{cases} \frac{1}{\sqrt{2}}, & p = 0, \\ 1, & 1 \leq p \leq 7. \end{cases} \tag{2.5}$$

Then, the quantization is done by multiplying each DCT coefficient matrix, $\boldsymbol{I[p, q]}$, with an quantization matrix that is determined by the compression quality factor ($\boldsymbol{Q[p, q]}$). Thus, the quantized coefficient matrix is given by:

$$\boldsymbol{I_{qt}[p, q]} = round(\frac{\boldsymbol{I[p, q]}}{\boldsymbol{Q[p, q]}}). \tag{2.6}$$

By computing the de-quantization and the inverse discrete cosine transform (IDCT):

$$\boldsymbol{I_{dqt}[p,q]} = \boldsymbol{I_{qt}[p,q]} \times \boldsymbol{Q[p,q]}, \tag{2.7}$$

the input image is reconstructed as:

$$j[x,y] = \sum_{p=0}^{7}\sum_{q=0}^{7} \frac{C[p]C[q]}{4} \boldsymbol{I_{dqt}[p,q]} \cos\frac{(2x+1)p\pi}{16} \cos\frac{(2y+1)q\pi}{16} \tag{2.8}$$

where $0 \leq x, y \leq 7$. Finally, the reconstructed input image is obtained when the above operations are applied to all of the image blocks.

The JPEG compression algorithm involves considerable multiplications with real numbers, which leads to the use of approximate floating-point arithmetic that provides acceptable quality loss. Although fixed-point arithmetic can be used in the encoding and decoding processes, additional effort and time are required for the implementation and the wider range of floating-point representation provides better quality image recovery. Therefore, approximate floating-point multipliers can be beneficial for improving the hardware efficiency of the JPEG compression algorithm.

2.3.2 Neural Networks

NNs are computational models that possess attractive characteristics of the biological NNs of the brain [38]. NNs perform tasks based on the machine learning algorithm that is able to learn from data. One of the most common tasks is classification, by which the input data is decided to belong to one among several possible categories. To solve the task, NNs process the data by artificial neurons and update the weights and biases through the training process consisting of feed-forward propagation and back propagation [39]. In the feed-forward process, the state of neurons in NNs is computed based on the output of neurons in the previous layer and is propagated to the following layer for further computation. As the basic unit of NNs, each neuron computes the sum of multiplication products of the inputs and their corresponding weights for the current layer. The sum is added with a bias and then passed to the activation function, which introduces non-linearity to the learning process. The basic computation for a neuron in the forward propagation can be expressed as:

$$a_k^l = \sigma(\sum_{i=1}^{n} w_{ik}^l x_i^{l-1} + b_k^l) \tag{2.9}$$

where a_k^l denotes the output value of the k-th neuron in the l-th layer, w_{ik}^l indicates the weight that connects the i-th neurons of the $(l-1)$-th layer, denoted by x_i^{l-1}, with the k-th neuron in the l-th layer, and b_k^l represents the bias value for the k-th neuron in the l-th layer. The sum is performed over the n neurons in the $(l-1)$-th layer. $\sigma()$ indicates the activation function that maps the sum obtained in the range from -1 to 1 or from 0 to 1 [39]. Particularly, for the input layer, x_i^0 represents the input data.

The outputs of neurons in the output layer are inspected by comparing with the target outputs. A loss function, denoted by L, is used to evaluate the training loss in the results of the feed-forward propagation. Since the training process aims to generate the weights and biases that are capable of performing the classification, the loss value needs to be minimized through the back-propagation. The gradient descent algorithm is usually adopted to minimize the loss function by scaling a small change in the weights and biases that lead to reduction in the loss value. The derivatives of L with respect to each weight and bias, i.e., $\frac{\partial L}{\partial w}$ and $\frac{\partial L}{\partial b}$, are computed first. For the k-th neuron in the l-th layer, the gradient of its output, denoted by δ_k^l, is essential for computing $\frac{\partial L}{\partial w}$ and $\frac{\partial L}{\partial b}$, given by:

$$\delta_k^l = \frac{\partial L}{\partial z_k^l} = \frac{\partial L}{\partial a_k^l}\sigma'(z_k^l) \tag{2.10}$$

where $z_k^l = \sum_{i=1}^{n} w_{ik}^l x_i^{l-1} + b_k^l$. δ_k^l is expressed differently for the output layer and the hidden layers. $\frac{\partial L}{\partial a_k^l}$ is computed directly with respect to the loss function for the output layer; however, when the l-th layer is a hidden layer, $\frac{\partial L}{\partial a_k^l}$ is computed using the gradients in the next following layers, e.g., the $(l+1)$-th layer. Thus, δ_k^l for a hidden layer is expressed as:

$$\delta_k^l = (\sum_{k=1}^{m} w_{kj}^{l+1}\sigma_j^{l+1})\sigma(z_k^l), \tag{2.11}$$

where w_{kj}^{l+1} is the weight that connects the k-th neurons in the l-th layer with the j-th neuron in the $(l+1)$-th layer. The sum is performed in the m neurons in the $(l+1)$-th layer. Thus, δ_k is computed recursively for each hidden layer in a backward direction. Therefore, the gradients of a weight and a bias, w_{ik}^{l} and b_{k}^{l}, are given by:

$$\frac{\partial L}{\partial w_{ik}^{l}} = \delta_k^l a_i^{l-1}, \tag{2.12}$$

$$\frac{\partial L}{\partial b_{k}^{l}} = \delta_k^l. \tag{2.13}$$

The computation shown above is for a single input sample. Depending on different gradient descent optimization methods, such as the mini-batch gradient descent [40], the stochastic gradient descent [41], among others, the weights and biases are updated using a learning rate (η) and their gradients obtained from the u input samples. Let $W_v^{'}$ and $B_v^{'}$ denote the gradients of the weights and biases for a single input sample in a NN model, the weights and biases are updated by:

$$W \rightarrow W - \frac{\eta}{u} \sum_{v=1}^{u} W_v^{'}, \tag{2.14}$$

$$B \rightarrow B - \frac{\eta}{u} \sum_{v=1}^{u} B_v^{'}, \tag{2.15}$$

The training process requires a massive number of MAC computations in the iterative forward-propagation and back-propagation algorithms. These delicate computations, especially for the gradients, are performed by using the floating-point representation due to the wider range of numeric precision compared to the fixed-point representation. Therefore, the use of hardware-efficient floating-point arithmetic is advantageous for the training of NNs.

Chapter 3

Representation, Formulation and Approximation for Multiplication

3.1 FP Representation and Multiplication

3.1.1 IEEE 754 FP Representation

The IEEE 754 standard defines the most commonly used formats for FP numbers. This format contains a 1-bit sign S, a w-bit exponent E and a q-bit mantissa M [42]. Fig. 3.1 shows the IEEE 754 representation of a single-precision FP number.

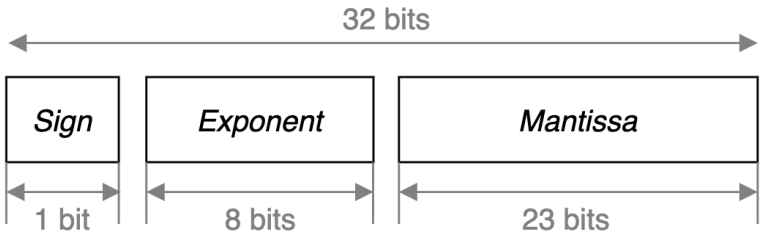


Figure 3.1: The IEEE-754 single-precision format.

In this format, a number N can be expressed in a base-2 scientific notation as follows:

$$N = (-1)^S \cdot 2^{E-bias} \cdot (1 + x), \tag{3.1}$$

where S is either 0 for a positive number or 1 for a negative number. To ensure unsigned integers in the exponent field, a bias $(2^{(w-1)} - 1)$, such as 127 for the single-precision, is added to the actual exponent value. $E - bias$ denotes the actual exponent

value of N. With the hidden '1', x is the fractional part of the FP number, and hence $0 \leq x < 1$.

3.1.2 Floating-Point Multiplication

In the IEEE 754 format, the FP multiplication involves three processes, including an XOR operation for the sign bits, the addition of the exponents, and the multiplication of the mantissa bits. Note that X is used to denote the actual mantissa, $1+x$, in the following formulation. Consider $P = A \times B$, which is computed as follows:

$$S_P = S_A \oplus S_B, \tag{3.2}$$

$$X_{AB} = (1 + x_A) \times (1 + x_B), \tag{3.3}$$

$$E_P = \begin{cases} E_A + E_B - bias, & X_{AB} < 2, \\ E_A + E_B - bias + 1, & otherwise, \end{cases} \tag{3.4}$$

$$X_P = \begin{cases} X_{AB}, & X_{AB} < 2, \\ X_{AB}/2, & otherwise, \end{cases} \tag{3.5}$$

where the sign bit, exponent and mantissa of A, B and P are respectively denoted with the corresponding subscripts. The exponent and mantissa of product P relate to the comparison of the obtained mantissa X_{AB} with 2. Note that (3.2) is valid for the sign computation of the proposed design, so it will not be discussed hereafter.

3.1.3 Logarithmic Floating-Point Multiplication

For $P = A \times B$ in the logarithm domain, the binary logarithmic multiplication converts the multiplication in (3.3) to an addition operation.

Assume that the logarithm, i.e., $\log_2(1 + x)$, is approximated by a term denoted by $\alpha(x)$, then $\log_2(X_{AB})$ is expressed as:

$$\log_2(X_{AB}) = \log_2(1 + x_A) + \log_2(1 + x_B) \cong \alpha(x_A) + \alpha(x_B). \tag{3.6}$$

Assume the anti-logarithm for $2^{(\alpha(x_A)+\alpha(x_B))}$ is approximated by a term denoted by $\beta(\alpha(x_A) + \alpha(x_B))$, then X_{AB} is expressed as:

$$X_{AB} \cong 2^{(\alpha(x_A)+\alpha(x_B))} \cong \beta(\alpha(x_A) + \alpha(x_B)). \tag{3.7}$$

Finally, E_P and X_P are obtained as in (3.4) and (3.5). Note that $\alpha()$ and $\beta()$ are used for ease of presentation.

3.2 Logarithmic Approximation and Mathematical Formulations for Multiplication

Since Mitchell's approximation method computes underestimated products that lead to error accumulation in the MAC output, two novel approximation methods that produce a double-sided error distribution are proposed.

3.2.1 Logarithmic Multiplication 1

Logarithmic FP Representation

The proposed approximation method-1 is based on the conversion of a logarithmic FP representation that differs from the IEEE 754 format. An FP number N is first converted into the format using its nearest power of two as per (2.3) and the corresponding mantissa. Since the IEEE 754 FP format provides the largest power of two smaller than N, by comparing the fraction x in (3.1) with 0.5, the nearest power of two can be determined for N.

If $x \geq 0.5$, N is closer to $2^{E-bias+1}$ than 2^{E-bias} (or equally away for the equal sign). The exponent E is incremented by 1 and, accordingly, the mantissa becomes $\frac{1+x}{2}$. In contrast, the exponent and mantissa of N remain the same as in (3.1) when $x < 0.5$. Let the converted exponent of N be denoted by $E^{'}$ and the converted mantissa by $X^{'}$. Then $E^{'}$ and $X^{'}$ are given by:

$$E^{'} = \begin{cases} E, & x < 0.5, \\ E+1, & x \geq 0.5, \end{cases} \tag{3.8}$$

$$X^{'} = 1 + x^{'} = \begin{cases} 1+x, & x < 0.5, \\ \frac{1+x}{2}, & x \geq 0.5, \end{cases} \tag{3.9}$$

where $0.75 \leq X^{'} < 1.5$.

Logarithm and Anti-logarithm Approximation

To better introduce the approximation method and its usage in multiplication, different notations are used. Consider the logarithm of $1 + k$, i.e., $\log_2(1 + k)$ and the anti-logarithm of l, i.e., 2^l.

Due to the conversion in representation as per (3.9), the logarithm is applied to X'. Thus the range of $1 + k$ is $[0.75, 1.5)$ and the logarithm approximation method-1 is applied over the domain $-0.25 \leq k < 0.5$. Since l is obtained as the sum of the two approximate logarithms, its range is $[-0.5, 1)$. Therefore, the anti-logarithm approximation method-1, as described in (2.2), is applied over the region of $-0.5 \leq l < 1$.

The logarithm approximation method-1 of the function $\log_2(1 + k)$ is shown in Fig. 3.2, where it is compared with Mitchell's method and the exact function. Note that method-1 computes the same underestimated results as Mitchell's method when both input operands are closer to 2^{E-bias}. However, the input operands closer to $2^{E-bias+1}$ are converted to overestimated logarithm values. As shown in Fig. 3.3, the anti-logarithm approximation method-1 produces the same overestimated results as Mitchell's method when l is in the range of $[0, 1)$, whereas it underestimates the results when l is negative. As a result, the proposed approximation method-1 can compute either underestimated or overestimated results depending on the input operands.

Approximation Error Analysis

Denote the error introduced in the logarithm approximation by $\varepsilon_1(k)$. Then $\varepsilon_1(k)$ for the proposed logarithm approximation method-1 is expressed as:

$$\varepsilon_1(k) = \log_2(1 + k) - k, \tag{3.10}$$

where $-0.25 \leq k < 0.5$. When $-0.25 \leq k < 0$, the approximate logarithm, i.e., $\alpha(k) = k$, is in the range of $-0.25 \leq k < 0$ (recall that $\alpha(k)$ denote the approximate result for computing the logarithm of k using a logarithm approximation), thus $\varepsilon_1(k)$

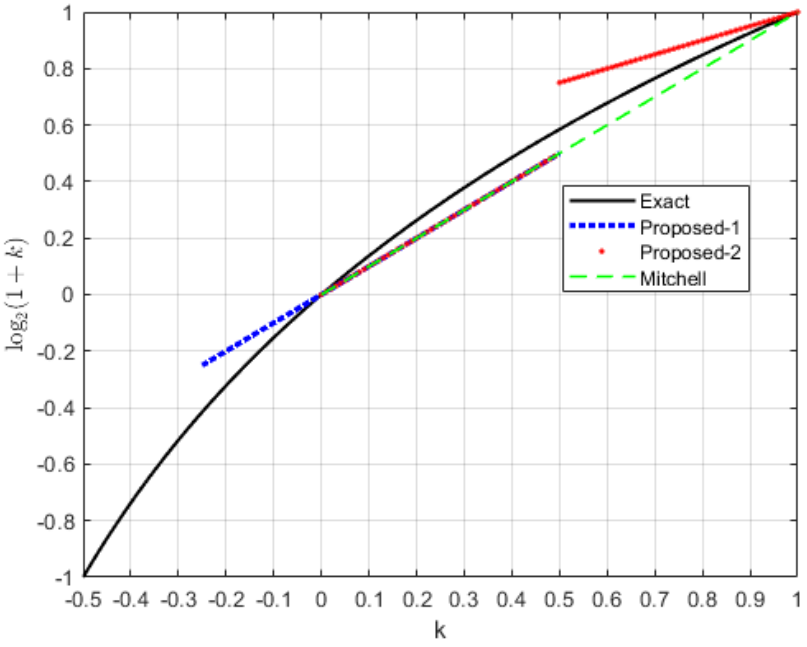


Figure 3.2: Approximations of $\log_2(1+k)$.

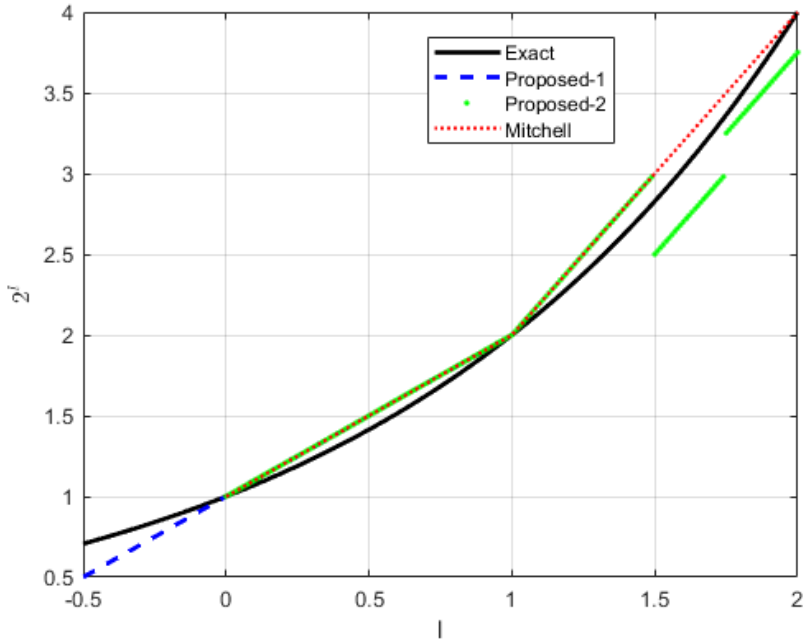


Figure 3.3: Approximations of 2^l.

is in the range of $[-0.1650, -0.0044)$, where the negative error indicates an overestimated result. When $0 \leq k < 0.5$, $0 \leq \alpha(k) < 0.5$), $\varepsilon_1(k)$ is within the range of $[0, 0.0860)$, where the positive error indicates an underestimated result.

When adding the approximate logarithms of k_i and k_j, i.e., $\alpha(k_i)$ and $\alpha(k_j)$, the accumulated errors can be reduced due to the double-sided error distribution for $\varepsilon_1(k)$. Consider $\varepsilon_1(k_i)$ and $\varepsilon_1(k_j)$ to be the logarithm approximation errors for k_i and k_j, respectively. There are three scenarios when adding $\alpha(k_i)$ and $\alpha(k_j)$, depending on the ranges of k_i and k_j:

- If $-0.25 \leq k_i, k_j < 0$, $-0.5 \leq \alpha(k_i) + \alpha(k_j) < 0$, then negative errors are accumulated, resulting in $-0.3300 \leq \varepsilon_1(k_i) + \varepsilon_1(k_j) < -0.0088$.
- If $0 \leq k_i, k_j < 0.5$, $0 \leq \alpha(k_i) + \alpha(k_j) < 1$, then $0 \leq \varepsilon_1(k_i) + \varepsilon_1(k_j) < 0.1720$ due to the accumulated positive errors.
- If $-0.25 \leq k_i(k_j) < 0$ and $0 \leq k_j(k_i) < 0.5$, then $-0.25 \leq \alpha(k_i) + \alpha(k_j) < 0.5$, and the range of $\varepsilon_1(k_i) + \varepsilon_1(k_j)$ is reduced to $[-0.1650, 0.0816)$.

Let $\varepsilon_2(l)$ denote the error introduced in the anti-logarithm approximation. Since $l = \alpha(k_i) + \alpha(k_j)$ according to (3.7), we have $-0.5 \leq l < 1$. Then $\varepsilon_2(l)$ for the proposed method-1 is given by:

$$\varepsilon_2(l) = 2^l - (l + 1). \tag{3.11}$$

Positive errors, i.e., $0.0030 \leq \varepsilon_2(l) < 0.2071$, are generated when $-0.5 \leq l < 0$, whereas non-positive errors are produced when $0 \leq l < 1$ since $-0.0860 \leq \varepsilon_2(l) < 0$.

Errors introduced in the logarithm conversion and the anti-logarithm conversion are accumulated, contributing to the overall approximation error, denoted by ε_{tot}. ε_{tot} is computed as:

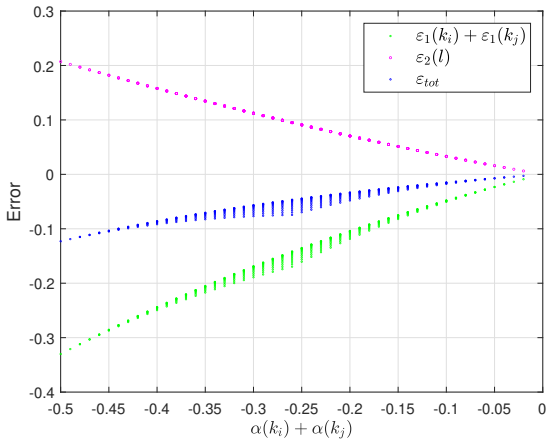
$$\varepsilon_{tot} = \varepsilon_1(k_i) + \varepsilon_1(k_j) + \varepsilon_2(l), \tag{3.12}$$

where $l = \alpha(k_i) + \alpha(k_j)$.

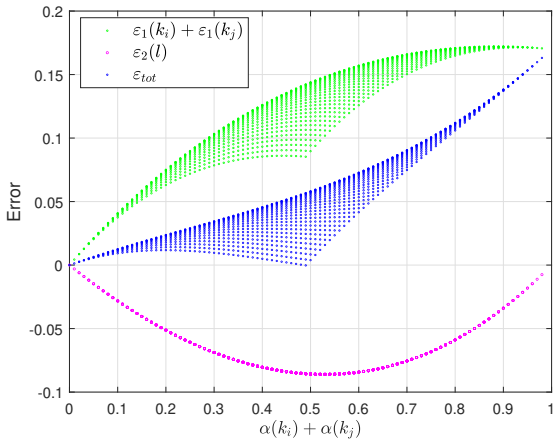
The sum of the two approximate logarithms, $\alpha(k_i) + \alpha(k_j)$, is involved in the two conversion processes, as it outputs from the logarithm conversion and is propagated as the input to the anti-logarithm conversion (l), according to (3.6) and (3.7). Therefore, ε_{tot} is analyzed with respect to $\alpha(k_i) + \alpha(k_j)$ to show the overall accumulation effect. For the proposed approximation method-1, depending on the range of $\alpha(k_i) + \alpha(k_j)$, ε_{tot} can be analyzed for the following three scenarios:

- When $-0.5 \leq \alpha(k_i)+\alpha(k_j) < 0$, $-0.3300 \leq \varepsilon_1(k_i)+\varepsilon_1(k_j) < -0.0088$, $0.0030 \leq \varepsilon_2(l) < 0.2071$.
- When $0 \leq \alpha(k_i)+\alpha(k_j) < 1$, $0 \leq \varepsilon_1(k_i)+\varepsilon_1(k_j) < 0.1720$, $-0.0860 \leq \varepsilon_2(l) < 0$.
- When $-0.25 \leq \alpha(k_i) + \alpha(k_j) < 0.5$, $-0.1650 \leq \varepsilon_1(k_i) + \varepsilon_1(k_j) < 0.0816$, $-0.0855 \leq \varepsilon_2(l) < 0.0908$.

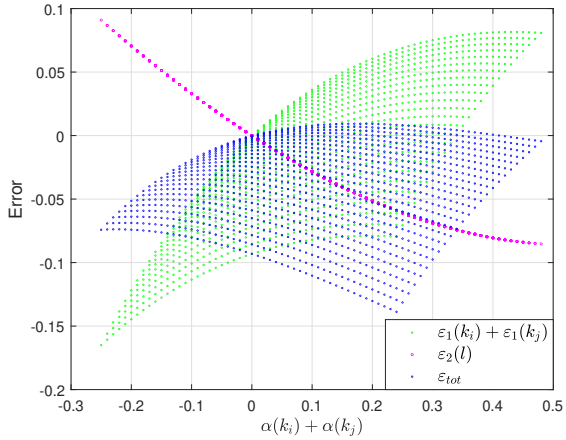
For all the three scenarios shown above, $\varepsilon_1(k_i) + \varepsilon_1(k_j)$, $\varepsilon_2(l)$, and ε_{tot} are plotted in Fig. 3.4 with respect to $\alpha(k_i) + \alpha(k_j)$. Note that k_i and k_j are sampled with a step size of 0.01. As shown in Fig. 3.4, the proposed approximation method-1 always produces errors with opposite signs in the two conversions, which largely reduces the accumulation effect. A value of $\alpha(k_i) + \alpha(k_j)$ can lead to multiple values of $\varepsilon_1(k_i) + \varepsilon_1(k_j)$ since multiple combinations of $\alpha(k_i)$ and $\alpha(k_j)$ can lead to the same sum. Fig. 3.5 shows the overall approximation error with respect to the four sub-regions that divide the range of $[-0.5, 1)$. The ε_{tot} under the first and third scenarios and under the second and third scenarios are averaged, respectively, for $-0.25 \leq \alpha(k_i) + \alpha(k_j) < 0$ and for $0 \leq \alpha(k_i) + \alpha(k_j) < 0.5$. As shown in Fig. 3.5, the overall approximation error is small, especially when $\alpha(k_i) + \alpha(k_j)$ is in the range of $[0, 0.5)$, due to the offset of errors with opposite signs in the two conversions. The approximation method-1 produces double-sided error distribution since ε_{tot} can be positive or negative.



(a) $-0.5 \leq \alpha(k_i) + \alpha(k_j) < 0$



(b) $0 \leq \alpha(k_i) + \alpha(k_j) < 1$



(c) $-0.25 \leq \alpha(k_i) + \alpha(k_j) < 0.5$

Figure 3.4: $\varepsilon_1(k_i) + \varepsilon_1(k_j)$, $\varepsilon_2(l)$, and ε_{tot} with respect to $\alpha(k_i) + \alpha(k_j)$ under the three situations for the proposed approximation method-1.

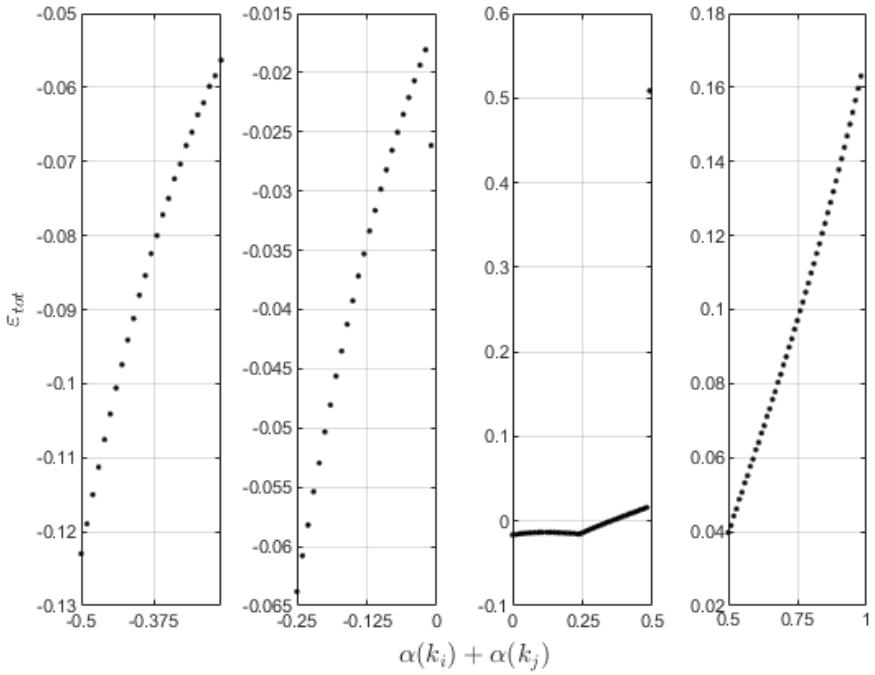


Figure 3.5: Overall approximation error of the logarithm and anti-logarithm conversions for the proposed approximation method-1.

Mathematical Formulations

For the logarithmic N using the logarithmic FP representation, the exponent is given as (3.8) and the logarithm is approximated as follows:

$$\log_2(X^{'}) = \log_2(1 + x^{'}) \cong \begin{cases} x, & x < 0.5, \\ \frac{1+x}{2} - 1, & x \geq 0.5. \end{cases} \tag{3.13}$$

For $P = A \times B$ in the logarithm domain, the exponent is still computed by addition, whereas the multiplication for the mantissa is converted to addition. Let $X^{'}_{AB} = X^{'}_{A} \times X^{'}_{B} = (1 + x^{'}_{A}) \times (1 + x^{'}_{B})$, and hence the logarithm of $X^{'}_{AB}$ is given by:

$$\log_2(X^{'}_{AB}) = \log_2(1 + x^{'}_{A}) + \log_2(1 + x^{'}_{B}) \cong \widehat{X}_A + \widehat{X}_B, \tag{3.14}$$

where $\widehat{X}_A$ and $\widehat{X}_B$ denote the approximate logarithms obtained by (3.13).

Using the anti-logarithm approximation method-1, $X^{'}_{AB}$ is obtained as:

$$X^{'}_{AB} \cong 2^{\widehat{X}_A + \widehat{X}_B} \cong 1 + \widehat{X}_A + \widehat{X}_B. \tag{3.15}$$

According to (3.13), we obtain $-0.5 \leq \widehat{X}_A + \widehat{X}_B < 1$, and thus, $0.5 \leq X'_{AB} < 2$. When $X'_{AB} < 1$ (or $\widehat{X}_A + \widehat{X}_B < 0$), X'_{AB} cannot be directly represented as the mantissa for the product P. In this case, X'_{AB} is multiplied by 2 and, accordingly, the exponent is reduced by 1.

Finally, the proposed logarithmic FP multiplication-1 is given by:

$$E_P = \begin{cases} E'_A + E'_B - bias, & \widehat{X}_A + \widehat{X}_B \geq 0, \\ E'_A + E'_B - bias - 1, & otherwise, \end{cases} \tag{3.16}$$

$$X_P = \begin{cases} 1 + \widehat{X}_A + \widehat{X}_B, & \widehat{X}_A + \widehat{X}_B \geq 0, \\ (1 + \widehat{X}_A + \widehat{X}_B) \times 2, & otherwise. \end{cases} \tag{3.17}$$

Note that E'_A and E'_B are the converted exponents, and $\widehat{X}_A$ and $\widehat{X}_B$ are the approximate logarithms given in the converted mantissas, for A and B, respectively.

3.2.2 Logarithmic Multiplication 2

Logarithm and Anti-logarithm Approximation

Consider a given number N, when $x \geq 0.5$, $\log_2 N = (E+1) + (\frac{1+x}{2} - 1)$ according to (3.13). By cancelling out the '+1' and '−1', we obtain $\log_2 N = E + \frac{1+x}{2}$, which leads to the logarithm approximation method-2 given by (3.18). Note that the conversions for the original exponent and the mantissa in (3.8) and (3.9) are avoided.

$$\log_2(1+k) \cong \begin{cases} k, & 0 \leq k < 0.5, \\ \frac{(1+k)}{2}, & 0.5 \leq k < 1, \end{cases} \tag{3.18}$$

where the range of the approximate logarithm is $[0, 1)$. Thus, the range of l is $[0, 2)$ (l is the sum of the two approximate logarithms). Consider using Mitchell's anti-logarithm method, as shown in Fig. 3.3, when $1.5 \leq l < 2$, larger approximate results will be produced in both the logarithm and anti-logarithm processes, which will lead to large error accumulation. Therefore, we propose to reduce the anti-logarithm value by subtracting certain values to ensure positive errors. The subtracted value is selected from our experiments for computing the overall accumulative error with various subtracted values. This process is shown in the following approximation error

analysis. 0.5 and 0.25 are used for the range of $[1.5, 1.75)$ and $[1.75, 2)$, respectively. The anti-logarithm approximation used in this method is given by:

$$2^l \cong \begin{cases} l+1, & l < 1, \\ 2 \times l, & 1 \leq l < 1.5, \\ 2 \times l - 0.5, & 1.5 \leq l < 1.75, \\ 2 \times l - 0.25, & 1.75 \leq l < 2, \end{cases} \tag{3.19}$$

The logarithm approximation method-2 is shown in Fig. 3.2. When the mantissas of both input operands are smaller than 0.5, method-2 computes the same underestimated logarithm approximation as Mitchell's method; otherwise, it computes overestimated results. As shown in Fig. 3.3, the same overestimated anti-logarithm approximation as Mitchell's method is produced when $0 \leq l < 1.5$, whereas underestimated anti-logarithm approximations are generated when $1.5 \leq l < 2$.

Approximation Error Analysis

$\varepsilon_1(k)$ for the proposed logarithm approximation method-2 is expressed as:

$$\varepsilon_1(k) = \begin{cases} \log_2(1+k) - k, & 0 \leq k < 0.5, \\ \log_2(1+k) - \frac{(1+k)}{2}, & 0.5 \leq k < 1, \end{cases} \tag{3.20}$$

where $0 \leq k < 1$. When $0 \leq k \in< 0.5$, the approximate logarithm, i.e., $\alpha(k) = k$, is in the range of $[0, 0.5)$, $\varepsilon_1(k)$ is in the range of $[0, 0.0860)$, where the positive errors indicate overestimated results. When $0.5 \leq k < 1$, the approximate logarithm, i.e., $\alpha(k) = \frac{(1+k)}{2}$, is in the range of $[0.75, 1)$, $\varepsilon_1(k)$ is within the range of $[-0.1650, -0.0022)$, where the negative errors indicate underestimated results.

Similar to the proposed method-1, the accumulation effect can be reduced when adding the approximate logarithms for k_i and k_j, i.e., $\alpha(k_i)$ and $\alpha(k_j)$, due to the double-sided error distribution. There are three scenarios when adding $\alpha(k_i)$ and $\alpha(k_j)$ depending on the ranges of k_i and k_j:

- If $0 \leq k_i, k_j < 0.5$, $0 \leq \alpha(k_i) + \alpha(k_j) < 1$, positive errors are accumulated, resulting in $0 \leq \varepsilon_1(k_i) + \varepsilon_1(k_j) < 0.1720$.

- If $0.5 \leq k_i, k_j < 1$, $1.5 \leq \alpha(k_i)+\alpha(k_j) < 2$, $-0.3300 \leq \varepsilon_1(k_i)+\varepsilon_1(k_j) < -0.0044$ due to the accumulated negative errors.

- If $0 \leq k_i(k_j) < 0.5$ and $0.5 \leq k_j(k_i) < 1$, $0.75 \leq \alpha(k_i) + \alpha(k_j) < 1.5$, the range of $\varepsilon_1(k_i) + \varepsilon_1(k_j)$ is reduced to $[-0.1650, 0.0838)$.

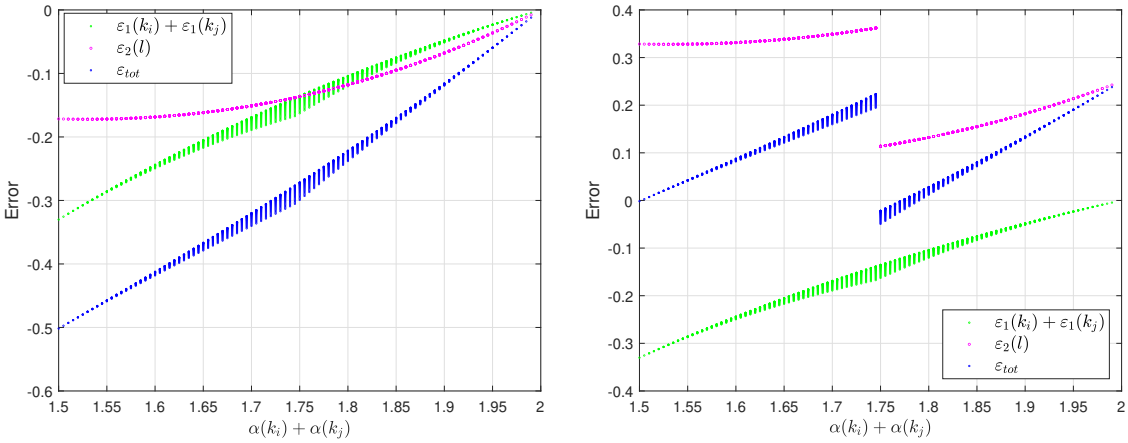
Using Mitchell's anti-logarithm approximation method, negative errors are generated both when $0 \leq l < 1$ and $1 \leq l < 2$ with $-0.0860 \leq \varepsilon_2(l) < 0$ and $-0.1721 \leq \varepsilon_2(l) < 0$, respectively. While analyzing ε_{tot} as per (3.12), we observed that, when $1.5 \leq \alpha(k_i)+\alpha(k_j) < 2$, the negative errors are accumulated in the two conversions due to $-0.3300 \leq \varepsilon_1(k_i)+\varepsilon_1(k_j) < -0.0044$ and $-0.1721 \leq \varepsilon_2(l) < -0.0076$. As shown in Fig. 3.6(a), ε_{tot} is plotted when $1.5 \leq \alpha(k_i) + \alpha(k_j) < 2$, which indicates the accumulation of negative errors leads to large ε_{tot}. Therefore, to reduce the accumulation effect for the range of $[1.5, 2)$, we propose to reduce the approximate anti-logarithm value by simply subtracting certain values from it, which ensures generating positive errors, i.e., $\varepsilon_2(l) > 0$. Denote the subtracted value by δ, the anti-logarithm approximation for the range of $[1.5, 2)$ is expressed as $2 \times l - \delta$ and $\varepsilon_2(l)$ is computed as:

$$\varepsilon_2(l) = 2^l - (2 \times l - \delta), \tag{3.21}$$

where $1.5 \leq l < 2$.

The value of δ is selected to reduce the error accumulation effect as much as possible while introducing a small hardware overhead. As shown in Fig. 3.6a, the absolute value of $\varepsilon_1(k_i) + \varepsilon_1(k_j)$ decreases with $\alpha(k_i) + \alpha(k_j)$ in the range of $[1.5, 2)$, indicating that δ should be decreasing with $\alpha(k_i)+\alpha(k_j)$ to produce the positive $\varepsilon_2(l)$ that can offset errors of the negative $\varepsilon_1(k_i) + \varepsilon_1(k_j)$ as much as possible. A trade-off is assessed when determining δ. Using different δ with excessively small intervals in the range of $[1.5, 2)$ can substantially increase the hardware overhead, whereas subtracting one single value in the range of $[1.5, 2)$ can be ineffectual or aggravate the error accumulation. We consider to use δ for the two sub-regions of $[1.5, 2)$ divided

by 1.75. By computing ε_{tot} with various values of δ, δ is determined to be 0.5 and 0.25 for the range of $[1.5, 1.75)$ and $[1.75, 2)$, respectively. Therefore, the proposed anti-logarithm approximation is given by (3.19).



(a) Using Mitchell's anti-logarithm approximation method.

(b) Using the proposed anti-logarithm approximation method.

Figure 3.6: $\varepsilon_1(k_i) + \varepsilon_1(k_j)$, $\varepsilon_2(l)$, and ε_{tot} for $1.5 \leq \alpha(k_i) + \alpha(k_j) < 2$.

$\varepsilon_2(l)$ for the proposed method-2 is given by:

$$\varepsilon_2(l) = \begin{cases} 2^l - (l+1), & 0 \leq l < 1, \\ 2^l - l \times 2, & 1 \leq l < 1.5, \\ 2^l - (2 \times l - 0.5), & 1.5 \leq l < 1.75, \\ 2^l - (2 \times l - 0.25), & 1.75 \leq l < 2, \end{cases} \tag{3.22}$$

where $0 \leq l < 2$. As shown in in Fig. 3.6b, for $1.5 \leq \alpha(k_i) + \alpha(k_j) < 2$, positive $\varepsilon_2(l)$ is produced to cancel out the negative $\alpha(k_i) + \alpha(k_j)$. ε_{tot} can then be analyzed for the following four scenarios:

- when $0 \leq \alpha(k_i) + \alpha(k_j) < 1$, $0 \leq \varepsilon_1(k_i) + \varepsilon_1(k_j) < 0.1720$, $-0.0860 \leq \varepsilon_2(l) < 0$.
- when $1.5 \leq \alpha(k_i) + \alpha(k_j) < 1.75$, $-0.3300 \leq \varepsilon_1(k_i) + \varepsilon_1(k_j) < -0.1418$, $0.3278 \leq \varepsilon_2(l) < 0.3603$.
- when $1.75 \leq \alpha(k_i) + \alpha(k_j) < 2$, $-0.135 \leq \varepsilon_1(k_i) + \varepsilon_1(k_j) < -0.0044$, $0.1135 \leq \varepsilon_2(l) < 0.2423$.

- when $0.75 \leq \alpha(k_i)+\alpha(k_j) < 1.5$, $-0.1650 \leq \varepsilon_1(k_i)+\varepsilon_1(k_j) < 0.0838$, $-0.1711 \leq \varepsilon_2(l) < 0$.

Negative errors are generated both when $0 \leq l < 1$ and $1 \leq l < 1.5$ with $-0.0860 \leq \varepsilon_2(l) < 0$ and $-0.1721 \leq \varepsilon_2(l) < 0$, respectively. When $1.5 \leq l < 1.75$ and $1.75 \leq l < 2$, positive errors are generated with $0.3278 \leq \varepsilon_2(l) < 0.3603$ and $0.1135 \leq \varepsilon_2(l) < 0.2423$, respectively. The overall approximation error is shown in Fig. 3.7 with respect to four sub-regions that divide the range of $[-0.5, 1)$. The ε_{tot} in the first and fourth scenarios are averaged for $0.75 \leq \alpha(k_i)+\alpha(k_j) < 1$. As shown in Fig. 3.7, a double-sided error distribution is generated by the approximation method-2.

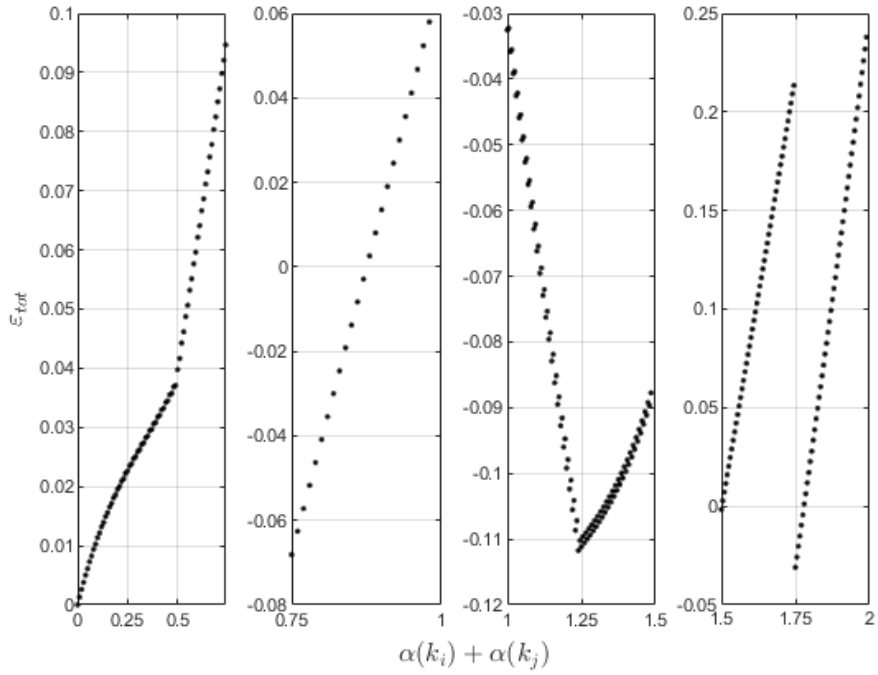


Figure 3.7: Overall accumulated errors of the logarithm and anti-logarithm conversions for the proposed approximation method-2.

Mathematical Formulations

Using the proposed logarithm approximation method-2, as per (3.18) and the anti-logarithm approximation, as per (3.19), X_{AB} is obtained. When $\widehat{X}_A + \widehat{X}_B \geq 1$, X_{AB} is divided by 2 to fit in the range of $[1, 2)$ and, accordingly, a carry to $E_A + E_B$ is

added to ensure the correct result.

Finally, the logarithmic FP multiplication in the proposed multiplication-2 is given by:

$$E_P = \begin{cases} E_A + E_B - bias + 1, & \widehat{X}_A + \widehat{X}_B \geq 1 \\ E_A + E_B - bias, & otherwise, \end{cases} \tag{3.23}$$

$$X_P = \begin{cases} 1 + \widehat{X}_A + \widehat{X}_B, & \widehat{X}_A + \widehat{X}_B < 1, \\ \widehat{X}_A + \widehat{X}_B, & 1 \leq \widehat{X}_A + \widehat{X}_B < 1.5, \\ \widehat{X}_A + \widehat{X}_B - 0.25, & 1.5 \leq \widehat{X}_A + \widehat{X}_B < 1.75, \\ \widehat{X}_A + \widehat{X}_B - 0.125, & 1.75 \leq \widehat{X}_A + \widehat{X}_B < 2. \end{cases} \tag{3.24}$$

3.3 Theoretical Error Analysis

The two proposed FP logarithmic multiplication methods introduce approximation errors in both the logarithm conversion and anti-logarithm conversion. The error distance is analyzed with the exponent ignored for simplicity.

According to (3.2)-(3.5), the result of exact multiplication is given by $P_{ex} = (1 + x_A) \times (1 + x_B) = 1 + x_A + x_B + x_A x_B$. The products for the proposed multiplication-1 and multiplication-2 algorithms, denoted as P_{ap1} and P_{ap2}, are given by:

$$P_1 = \begin{cases} 1 + x_A + x_B, & x_A, x_B < 0.5, & \text{(3.25a)} \\ 1 + 2x_A + x_B, & x_A < 0.5, x_B \geq 0.5, & \text{(3.25b)} \\ 1 + x_A + 2x_B, & x_A \geq 0.5, x_B < 0.5, & \text{(3.25c)} \\ 2x_A + 2x_B, & x_A, x_B \geq 0.5, & \text{(3.25d)} \end{cases}$$

$$P_2 = \begin{cases} 1 + x_A + x_B, \quad x_A, x_B < 0.5, & (3.26a) \\ \dfrac{3 + 2x_A + x_B}{2} \text{ or } 1 + 2x_A + x_B, & \\ \qquad\qquad x_A < 0.5, x_B \geq 0.5, & (3.26b) \\ \dfrac{3 + x_A + 2x_B}{2} \text{ or } 1 + x_A + 2x_B, & \\ \qquad\qquad x_A \geq 0.5, x_B < 0.5, & (3.26c) \\ 1.5 + x_A + x_B \text{ or } 1.75 + x_A + x_B, & \\ \qquad\qquad x_A, x_B \geq 0.5. & (3.26d) \end{cases}$$

By comparing the equations under different conditions, the error, $Err = P_{ex} - P_{ap}$, can be derived as follows:

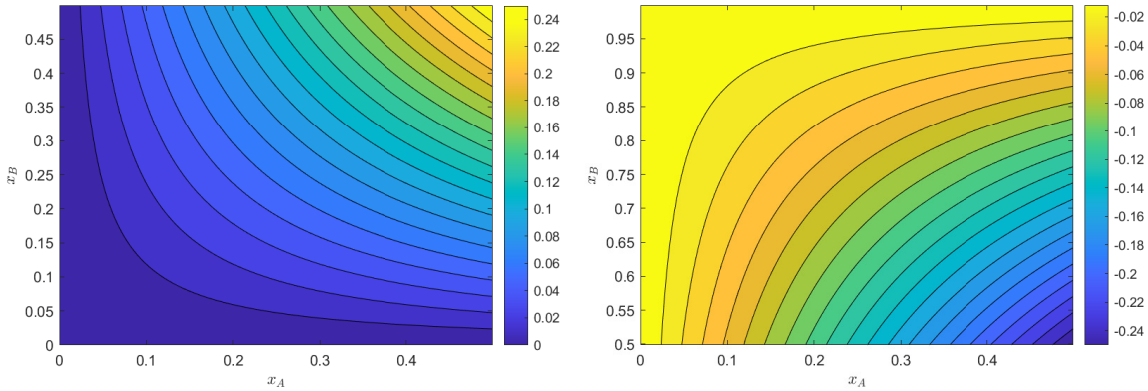
$$Err_1 = \begin{cases} x_A x_B, \quad x_A, x_B < 0.5, & (3.27a) \\ x_A(x_B - 1), \quad x_A < 0.5, x_B \geq 0.5, & (3.27b) \\ x_B(x_A - 1), \quad x_A \geq 0.5, x_B < 0.5, & (3.27c) \\ (1 - x_A)(1 - x_B), \quad x_A, x_B \geq 0.5, & (3.27d) \end{cases}$$

$$Err_2 = \begin{cases} x_A x_B, \quad x_A, x_B < 0.5, & (3.28a) \\ \dfrac{x_B + 2x_A x_B - 1}{2} \text{ or } x_A(x_B - 1), & \\ \qquad\qquad x_A < 0.5, x_B \geq 0.5, & (3.28b) \\ \dfrac{x_A + 2x_A x_B - 1}{2} \text{ or } x_B(x_A - 1), & \\ \qquad\qquad x_A \geq 0.5, x_B < 0.5, & (3.28c) \\ x_A x_B - 0.5 \text{ or } x_A x_B - 0.75, & \\ \qquad\qquad x_A, x_B \geq 0.5. & (3.28d) \end{cases}$$

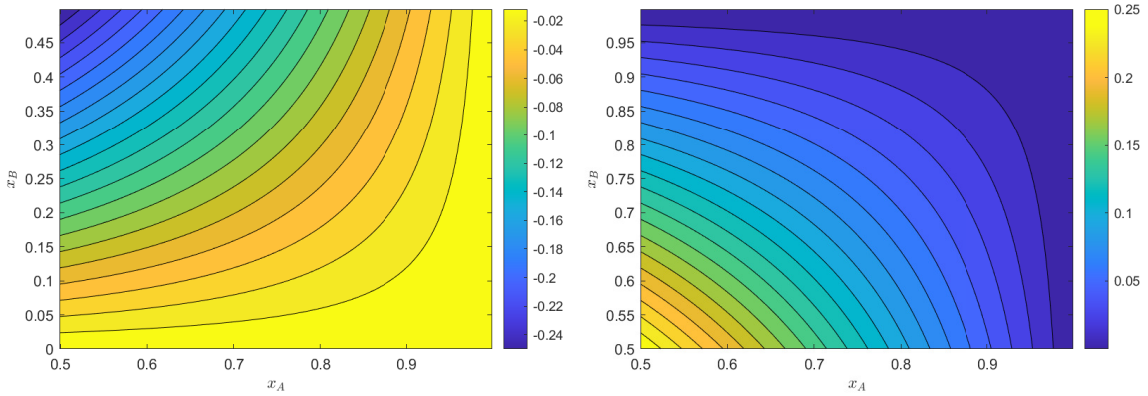
The contour maps of Err_1 and Err_2 are shown in Fig. 3.8 and Fig. 3.9, respectively. The largest positive Err_1 is 0.25 when both x_A and x_B are 0.5, as shown in Fig. 3.8(d),

while the negative Err_1 that has the maximum absolute value is close to -0.25 when either one of x_A and x_B approaches 0.5 while the other one equals 0.5, as seen in Fig. 3.8(b) and Fig. 3.8(c). Err_1 is larger than 0 when $x_A, x_B < 0.5$ or $x_A, x_B \geq 0.5$ and it is less than 0 under the other two conditions. The negative Err_2 has the largest absolute value of -0.5 when both x_A and x_B are 0.5, while the largest positive Err_2 is close to 0.5 when both x_A and x_B approach 1. Specifically, When $x_A, x_B < 0.5$, Err_2 is the same as Err_1 as shown in Fig. 3.8(a). For each of the other three scenarios as shown in (3.28b)-(3.28d), Err_2 is given by two expressions. Fig. 3.9(a) and Fig. 3.9(b) present the contour maps for the first equation of (3.28b) and (3.28c), respectively. In these two cases, the largest positive Err_2 is very close to 0.5 when one of x_A and x_B approaches 0.5 and another one approaches 1; the negative Err_2 with the maximum absolute value is -0.25 when one of x_A and x_B is 0 and the other is 0.5. Err_2 is expressed by the second equations of (3.28b) and (3.28c), the same as the multiplication-1, as shown in Fig. 3.8(b) and Fig. 3.8(c). When $x_A, x_B \geq 0.5$, the contour maps for Err_2 are presented in Fig. 3.9(c) and Fig. 3.9(d).

Therefore the maximum $|Err|$ is 0.25 and 0.5 for the multiplication-1 and the multiplication-2, respectively. In comparison, the LAM [3] that uses Mitchell's approximation has the largest absolute error of 0.25 and always underestimates the product. However, the average errors for the multiplication-1 and the multiplication-2 can be reduced since the positive errors and negative errors tend to cancel each other in a sum of products.

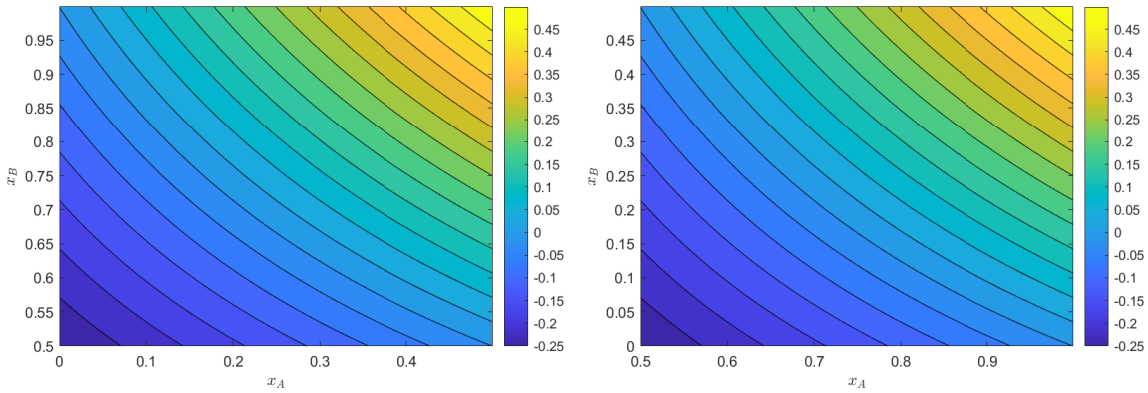


(a) $x_A, x_B < 0.5$ (b) $x_A < 0.5, x_B \geq 0.5$



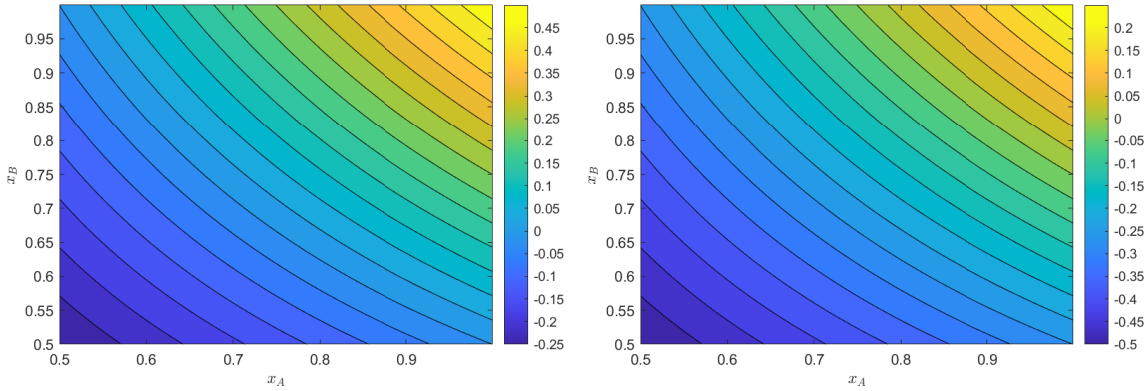
(c) $x_A \geq 0.5, x_B < 0.5$ (d) $x_A, x_B \geq 0.5$

Figure 3.8: The contour maps of the theoretical error (error = exact - approximate) with respect to x_A and x_B for the multiplication-1 algorithm.



(a) $x_A < 0.5, x_B \geq 0.5$, $\frac{x_B+2x_Ax_B-1}{2}$

(b) $x_A \geq 0.5, x_B < 0.5$, $\frac{x_A+2x_Ax_B-1}{2}$



(c) $x_A, x_B \geq 0.5$, $x_Ax_B - 0.5$

(d) $x_A, x_B \geq 0.5$, $x_Ax_B - 0.75$

Figure 3.9: The contour maps of the theoretical error (error = exact - approximate) with respect to x_A and x_B for the multiplication-2 algorithm.

Chapter 4

Circuit Designs of Floating-Point Logarithmic Multipliers

In this chapter, a generic circuit architecture is first introduced for the proposed FP LMs. The circuit design for these multipliers is then presented in detail. Lastly, the radix-4 logarithm (R4L) is considered to further reduce the hardware cost of the proposed designs.

4.1 A Generic Circuit Architecture

Fig. 4.1 presents the generic circuit architecture for the proposed FP LMs according to their mathematical formulations. Taking advantage of the IEEE 754 FP format, the sign S, the exponent E, and the mantissa M of the FP number can be obtained directly. The w-bit E is given by $E[w-1]E[w-2]\cdots E[0]$. $1.M$ is used to denote the actual mantissa. Here, M contains q bits, e.g., 23 bits for single-precision, after the binary point and the hidden '1', and is given as $M[q-1]M[q-2]\cdots M[1]M[0]$. Note that $1.M$ represents $1+x$ in (3.1).

The sign bit of the product, S_P, is obtained using an XOR gate with two input signs, S_A and S_B. For the multiplier that uses the approximation method-1, the exponents (E_A and E_B) and the mantissas ($1.M_A$ and $1.M_B$) are converted into the nearest-one FP representation, whereas for the approximation method-2, the conversion is not required. The converted exponents or original exponents serve as the

inputs of the adder, denoted by E'_A and E'_B, and the sum is denoted by E'_P. The converted mantissas are propagated to the logarithm approximation block to compute the approximate logarithms, denoted as M'_A and M'_B. Since the multiplication is replaced by the addition operation in the logarithm domain, M'_A and M'_B are added to obtain the sum, denoted by M'_P, which is then used in the anti-logarithm approximation block to compute its anti-logarithm. In the following adjustment block, the approximate anti-logarithm will be normalized to the range of $[1, 2)$ if needed, and E'_P will be adjusted accordingly, resulting in E_P and M_P. E_P is added with the bias to comply with the IEEE standard and any exception (such as overflow, underflow, and "not a number") is reported. The two inputs are checked for exceptions at the beginning of the computation. Note that the rounding unit is not required in the inexact design since it is already inherently imprecise.

For the proposed multiplier designs, instead of being implemented directly according to the equations, the circuits are simplified to reduce the hardware complexity and latency. Specifically, some arithmetic operations are replaced with simpler operations. By merging multiple computations, the circuit blocks in the same color, as shown in Fig. 4.1, are integrated into a single block. In particular, the representation conversion blocks of the exponent and the mantissa are shown in dashed boxes since the conversion is not always required.

4.2 FP Logarithmic Multiplier-1

4.2.1 Logarithm approximation and addition of approximate logarithms

As shown in Fig. 4.2, for the two given mantissas, M_A and M_B are used as the inputs for the two FP logarithm estimators (FP-LEs), which compute the approximate logarithm of the mantissa. For FPLM-1, both the representation conversion and logarithm approximation for the mantissa are implemented by an FP-LE. Simple wire routing is used to implement the FP-LE, as shown in Fig. 4.2. The nearest power

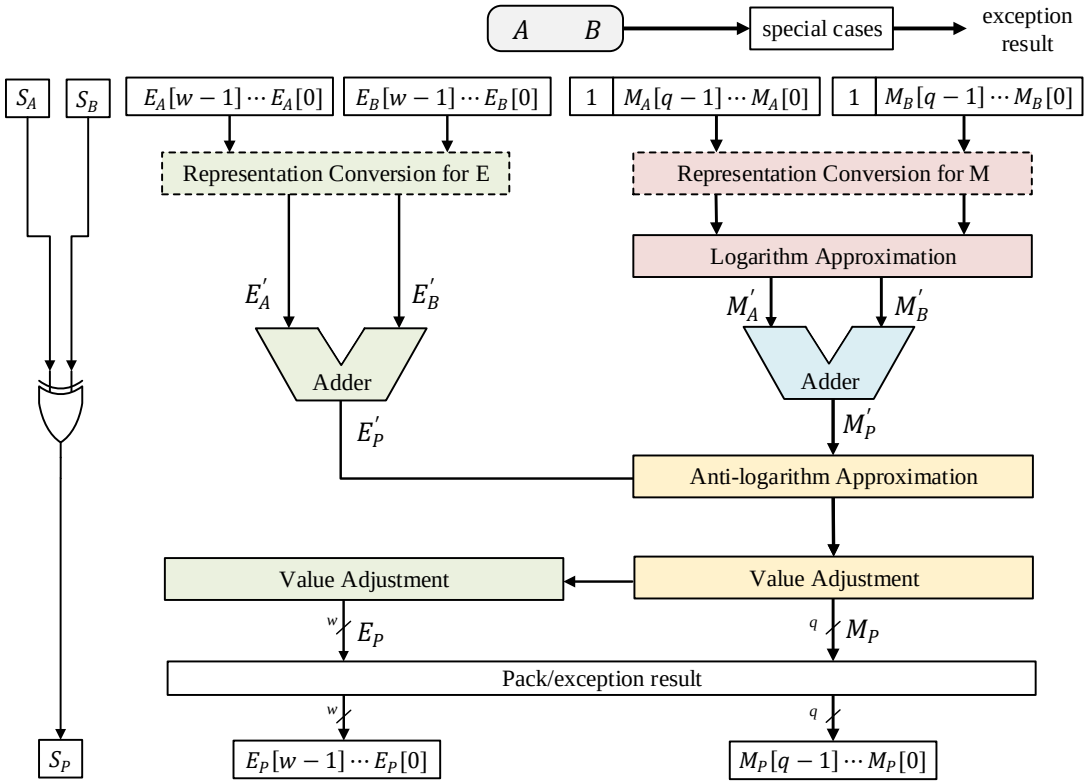


Figure 4.1: The generic circuit architecture for proposed designs.

of two for each FP number can be determined by simply checking the leading bit of the explicit mantissa (without the hidden 1), $M[q-1]$, and hence a 2-to-1 multiplexer is used to obtain the approximate logarithm, M'. According to (3.13), the left shifter for implementing the division by 2 is replaced by wire routing. Therefore, $(1+x)/2$ is implemented as $0.1M[q-1]\cdots M[1]$. When $M[q-1]=1$, the approximate logarithm, $(1+x)/2-1$, is obtained as a negative number in 2's complement, i.e., $M' = 1.1M[q-1]\cdots M[1]$; otherwise, M' is $0.M[q-1]\cdots M[0]$. In the case of $M[q-1]=1$, the LSB of M', $M[0]$, is discarded to keep M' in $q+1$ bits.

M'_A and M'_B are then summed using a $q+1$-bit adder.

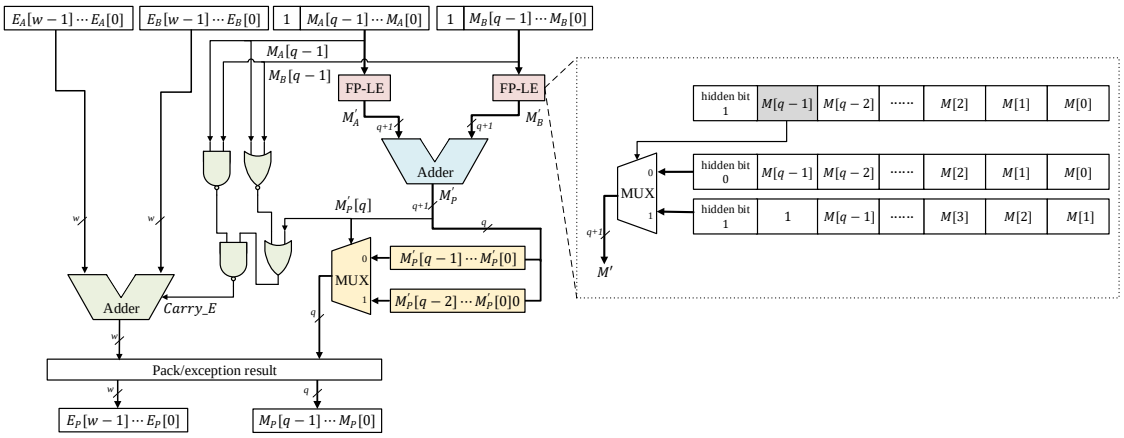


Figure 4.2: The circuit design of the first proposed floating-point logarithmic multiplier, FPLM-1.

4.2.2 Anti-logarithm approximation and value adjustment

The anti-logarithm approximation and value adjustment are implemented together using a multiplexer, as shown in Fig. 4.2. The $q+1$-bit M'_P obtained from the adder is the input for this block. For FPLM-1, as per (3.17), since $\widehat{X}_A + \widehat{X}_B$ can be negative in some cases, the MSB of M'_P, i.e., $M'_P[q]$ is the sign bit of M'_P in 2's complement. When $\widehat{X}_A + \widehat{X}_B < 0$, $M'_P[q] = 1$; otherwise, $M'_P[q] = 0$. Therefore, $M'_P[q]$ is used as the selection signal for the multiplexer. $1 + \widehat{X}_A + \widehat{X}_B$ is implemented as $0.M'_P[q-1]\cdots M'_P[0]$ or $1.M'_P[q-1]\cdots M'_P[0]$ when $M'_P[q] = 1$ or $M'_P[q] = 0$, respectively. According to (3.17), $-0.5 \leq \widehat{X}_A + \widehat{X}_B < 1$, meaning that $M'_P[q-1] = 1$ when $M'_P[q] = 1$. Therefore, in the case of $M'_P[q] = 1$, $1.M'_P[q-2]\cdots M'_P[0]0$ is obtained by performing the $\times 2$ operation on $0.M'_P[q-1]\cdots M'_P[0]$.

4.2.3 Addition of exponents and value adjustment

The circuits for the exponent conversion, addition and value adjustment, shown as the green blocks in Fig. 4.1, are implemented together by integrating and simplifying multiple computations. For FPLM-1, according to (3.8), the exponents of the two operands are converted first, practically depending on $M_A[q-1]$ and $M_B[q-1]$. Then the converted exponents are added subsequently to obtain E'_P, which is subtracted by 1 if $M'_P[q] = 1$, as per (3.16). The required circuits to implement these operations are simplified to one adder with a carry-in bit, $Carry\_E$, that is determined by the modified value of E_A and E_B. When $M_A[q-1]M_B[q-1]$ are '00' or '11', $M'_P[q]$ can only be '0' or '1', respectively; otherwise, $M'_P[q]$ can be '0' or '1' in either case. According to (3.13) and (3.16), when both mantissa values are smaller than 0.5 ($M_A[q-1]M_B[q-1] = 00$), $\widehat{X}_A + \widehat{X}_B \geq 0$ ($M'_P[q] = 0$), which means $E'_A + E'_B$ is not modified; hence, $Carry\_E = 0$. When both mantissas are greater than or equal to 0.5 ($M_A[q-1]M_B[q-1] = 11$), $\widehat{X}_A + \widehat{X}_B < 0$ ($M'_P[q] = 1$), which means that $E'_A + E'_B$ is subtracted by 1, i.e., $E_A + 1 + E_B + 1 - 1 = E_A + E_B + 1$; therefore, $Carry\_E = 1$.

Therefore, the $Carry\_E$ for FPLM-1 is obtained by using (4.1). As shown in Fig.

4.2, four logic gates are used to generate $Carry\_E$.

$$Carry\_E = \overline{\overline{(\,M'_P[q] + \overline{(\,M_A[q-1] + M_B[q-1]\,)}\,)}} \cdot \overline{\overline{(\,M_A[q-1] \cdot M_B[q-1]\,)}}\,. \tag{4.1}$$

4.3 FP Logarithmic Multiplier-2

4.3.1 Logarithm approximation and addition of approximate logarithms

As shown in Fig. 4.3, for FPLM-2, the FP-LE implements the logarithm approximation for the mantissa without the conversion to the nearest-one FP representation. As per (3.18), $(1 + x)/2$ is implemented as $0.1M[q-1]\cdots M[1]$ when $M[q-1] = 1$, by performing the division by 2 operation with wire routing; x is obtained as $0.M[q-1]\cdots M[0]$ when $M[q-1] = 0$. In both cases, the hidden bit is 0, which has no effect on the addition result. Thus, M' is obtained without the hidden bit and a q-bit adder is used for the addition of M'_A and M'_B with a carry-out signal, C_{out}, as shown in Fig. 4.3. In the case of $M[q-1] = 1$, the LSB of M', $M[0]$, is discarded to keep M' as q bits.

4.3.2 Anti-logarithm approximation and value adjustment

According to (3.24), $x'_A+x'_B$, is obtained as $C_{out}.M'_P[q-1]\cdots M'_P[0]$. When $x'_A+x'_B < 1$, which means $C_{out} = 0$, the mantissa, $1+x'_A+x'_B$, is obtained as $1.M'_P[q-1]\cdots M'_P[0]$. When $1 \le x'_A + x'_B < 1.5$, meaning that $C_{out}M'_P[q-1] = 10$, $x'_A + x'_B$ is also obtained as $1.M'_P[q-1]\cdots M'_P[0]$. When $1.5 \le x'_A + x'_B < 1.75$, $C_{out}M'_P[q-1]M'_P[q-2] = 110$, $x'_A + x'_B - 0.25$ is implemented by $1.01M'_P[q-3]\cdots M'_P[0]$. When $1.75 \le x'_A + x'_B < 2$, $C_{out}M'_P[q-1]M'_P[q-2] = 111$, $x'_A + x'_B - 0.125$ is implemented by $1.101M'_P[q-4]\cdots M'_P[0]$ or $1.110M'_P[q-4]\cdots M'_P[0]$ if $M'_P[q-3]$ is 0 or 1, respectively. Therefore, $M_P[q-4]\cdots M_P[0]$ is obtained as $M'_P[q-4]\cdots M'_P[0]$ and $M_P[q-1]\cdots M_P[q-3]$ is determined by $M'_P[q-1]\cdots M'_P[q-3]$ and C_{out}. The circuit for computing M_P is

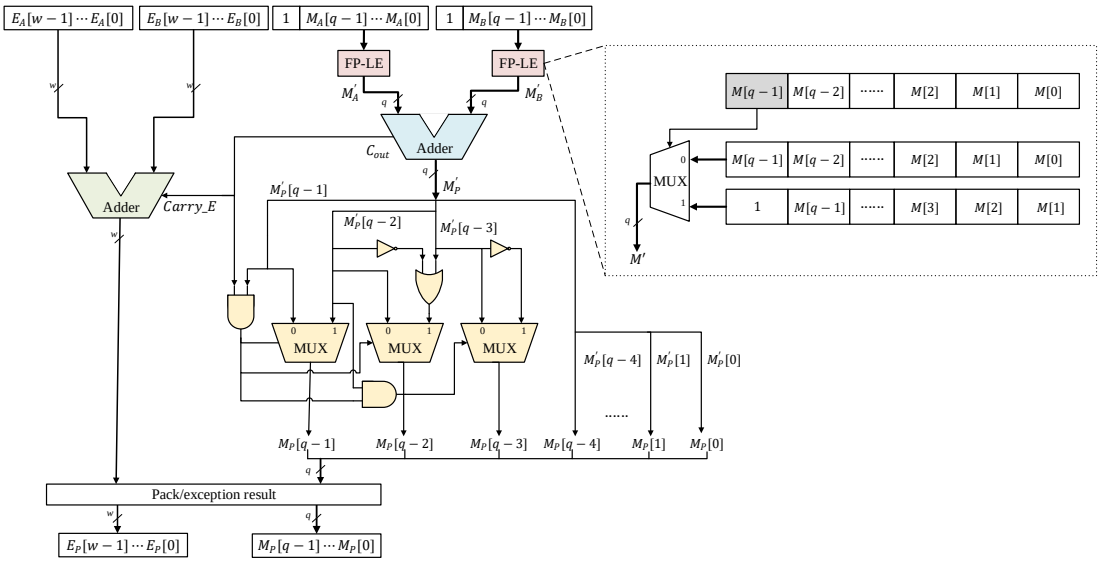


Figure 4.3: The circuit design of the second proposed floating-point logarithmic multiplier, FPLM-2.

shown in Fig. 4.3.

4.3.3 Addition of exponents and value adjustment

For FPLM-2, according to (3.23), E_A and E_B are summed and increased by 1 if $C_{out} = 1$. Therefore, as shown in Fig. 4.3, C_{out} is directly used to generate $Carry\_E$.

4.4 Designs using the Radix-4 Logarithm

The radix-4 logarithm (R4L) was introduced to reduce the hardware complexity of the FP logarithm multipliers by exploiting the relation between the base 2 and base 4 logarithms. The base 2 logarithm of a given number N, is two times its base 4 logarithm, i.e., $\log_2 N = 2\log_4 N$. This indicates that the base 4 logarithm can be stored with a smaller bit-width than the base 2 logarithm in a hardware implementation. Therefore, we propose to convert the approximate logarithm of the mantissa to a radix-4 logarithm in an intermediate computation process to reduce the hardware cost of an FP LM. The radix-4 logarithm conversion is only used for the mantissa since applying to the exponent leads to large errors.

The R4L approach is specified in Algorithm 1. The approximate logarithms for the two mantissas, denoted by m_A and m_B, are obtained by applying the logarithm approximation to the mantissa that is concatenated with the hidden '1'. Note that the & operator indicates concatenation. m_A and m_B are converted to the radix-4 logarithm by simply discarding their LSB, resulting in $m_A[q:1]$ and $m_B[q:1]$. The bit-width of the resulting approximate logarithm is reduced, which leads to a smaller hardware cost for the adder. Then the result, sum, is concatenated with '0' to ensure the correctness of the anti-logarithm approximation. When q is 1, the addition can be implemented with just an OR gate. The output, M_P, is obtained after normalizing the anti-logarithm result to make it lie in the range of $[0, 1)$.

The R4L approach was applied to the proposed FPLM-1, FPLM-2, and the conventional logarithmic multiplier that uses Mitchell's approximation [21] to reduce their

Algorithm 1 A Radix-4 Logarithm (R4L) Approach

Require: Mantissas: M_A and M_B; Bit-width of mantissa: q; Logarithm approximation method: $\alpha()$; Anti-logarithm approximation method: $\beta()$;
Ensure: Mantissa of the product: M_P.

```
if q ≥ 1 then
    m_A ← α('1' & M_A)
    m_B ← α('1' & M_B)
    if q > 1 then
        sum ← ADD(m_A[q : 1], m_B[q : 1])
    else
        sum ← OR(m_A[1], m_B[1])
    end if
    h ← β(sum & '0')
    M_P ← NORM(h)
else
    M_P ← 0
end if
```

hardware complexity. The obtained multiplier designs are denoted as FPLM-1-r4, FPLM-2-r4, and CLM-r4.

4.5 Optimization for Reduced Bit-Width in the Mantissa

With a reduced mantissa bit-width, the circuit can be further optimized. Consider an FP number with the FP8 format, which has a 2-bit mantissa. Its mantissa can only represent four binary numbers, i.e., 00, 01, 10, and 11.

For FPLM-1, the circuits for the FP-LE and the anti-logarithm approximation with value adjustment can be simplified. According to (3.13), M' can be given by:

$$M' = \{M[1],\ M[1],\ \overline{M[1]} \cdot M[0] + M[1]\}, \tag{4.2}$$

where the braces represent concatenation.

Accordingly, M_P is expressed as:

$$M_P = \{\overline{M'_P[2]} \cdot M'_P[1] + M'_P[2] \cdot M'_P[0],\ \overline{M'_P[2]} \cdot M'_P[0]\}. \tag{4.3}$$

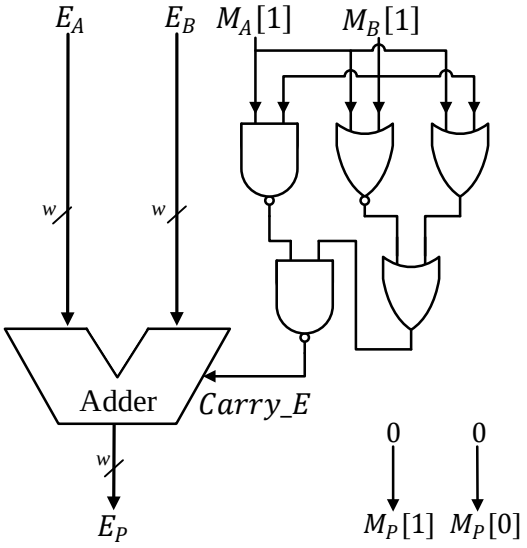
For FPLM-2, the circuits of the FP-LE and the anti-logarithm approximation with value adjustment, are simplified in the FP8 format. According to (3.18), M' is given by:

$$M' = \{M[1],\ M[1] + M[0]\}. \tag{4.4}$$

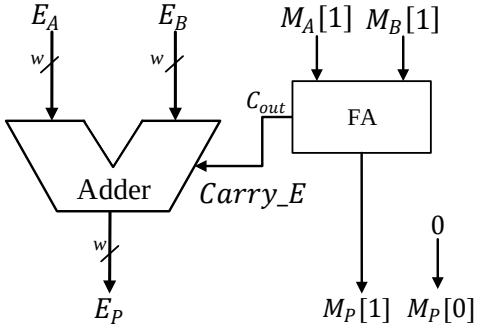
The 2-bit M'_P and C_{out}, obtained from the addition, compute M_P as per (3.19). M_P for FPLM-2 in the FP8 format is then expressed as:

$$M_P = \{\overline{C_{out} \cdot M'_P[1]} \cdot M'_P[1],\ \overline{C_{out} \cdot M'_P[1]} \cdot M'_P[0] + C_{out} \cdot M'_P[1]\}. \tag{4.5}$$

For the reduced mantissa bit-width, FPLM-1-r4, FPLM-2-r4, and CLM-r4 are also simplified, as shown in Fig. 4.4. Particularly, for FPLM-1-r4, when both $M_A[1]$ and $M_B[1]$ are 1, M_P along with its hidden bit is obtained as 0. Compared with the result of M_P obtained without using the R4L approach, an error of -1 is produced in this case. This error is eliminated by simply not clearing the hidden bit of M_P to 0. No extra circuit is required since M_P is computed directly as the explicit mantissa with the hidden bit 1. As shown in Fig. 4.4a, since M_P is computed independently, the adder is saved for FPLM-1-r4. The simplified circuit designs for FPLM-2-r4 and CLM-r4 of FP8 format are the same, as shown in Fig. 4.4b.



(a) FPLM-1-r4.



(b) FPLM-2-r4 and CLM-r4.

Figure 4.4: The simplified multiplier designs for the FP8 format.

Chapter 5

Performance Evaluation

In this chapter, the performance of the five proposed multipliers (FPLM-1, FPLM-2, FPLM-1-r4, FPLM-2-r4, and CLM-r4) is evaluated by comparing them with the conventional FP multiplier (FPM) and LAM [3], in terms of accuracy and circuit performance.

5.1 Accuracy Evaluation

5.1.1 Error Assessment

Two error metrics are considered to evaluate the error characteristics of the proposed FP LMs.

- The mean relative error distance (MRED) is the average value of all possible relative absolute error distances.
- The average error (AE) is the average difference that can be positive or negative between the exact and approximate products.

Four FP precision levels are considered for the evaluation of each multiplier: 32-bit single-precision, 16-bit half-precision, Brain FP (bfloat16) and 8-bit FP (FP8) format. The FP8 is chosen in the form of (1, 5, 2) bits for the sign, exponent and mantissa since it was found to perform the best with respect to classification accuracy after the simulation of using various FP8 formats [20]. The product obtained by the

single-precision exact FP multiplier is used as the benchmark since the truncation of mantissa bits inherently introduces errors for the 16-bit and 8-bit implementations. A sample of 10^7 random cases from uniform and standard normal distributions (μ=0 and σ=1) were generated to obtain the results in Table 5.1. Note that the uniformly distributed random cases are generated over the interval of $[1, 2)$ for assessing the errors introduced in the mantissa computation.

The results in Table 5.1 show that, for the single- and half-precisions, the FPLM-1 is the most accurate design with the lowest MRED and |AE| for both two distributions. For the bfloat16 format, FPLM-2-r4 achieves the smallest |AE| with respect to the uniform distribution, while FPLM-1 is the most accurate for MRED. Compared to the LAM, FPLM-1 performs up to 30.8% more accurately for the bfloat16 format with respect to MRED, and achieves a smaller |AE| by up to 2.58×10^3 times for single-precision. FPLM-2 achieves up to 20.1% smaller MRED and 70.5% smaller |AE| for the bfloat16 format. The smaller |AE|s show the advantage of the double-sided error distribution obtained for the proposed designs. Note that for the 32-bit precision, the error metrics that are shown as identical have a difference less than 10^{-6}. For the 16-bit precision, FPLM-1-r4 and CLM-r4 produce larger errors than FPLM-1 and LAM, respectively, whereas FPLM-2-r4 is slightly more accurate than FPLM-2. This is due to the relation between its error distribution and the bit-width, as explained in the next section. For FP8, all of the multipliers produce large errors. FPLM-2 is the most accurate multiplier, achieving up to a 16.1% smaller MRED and up to 45.9% smaller |AE| than LAM.

5.1.2 The Relation between FP Precisions and Error Behavior

According to Table 5.1, the comparison results are the same for 32- and 16-bit precisions, while being different for the FP8 format. Fig. 5.1 shows the MREDs obtained when varying the mantissa width for FP LMs and the FPM with the single-precision.

Table 5.1: Error Metrics for the Multipliers

Multipliers	Uniform Distribution		Normal Distribution	
Single-Precision				
	MRED	|AE|	MRED	|AE|($\times 10^{-5}$)
FPLM-1	**0.0288**	$\mathbf{3.2 \times 10^{-5}}$	**0.0288**	**0.83**
FPLM-2	0.0368	0.0416	0.0373	1.09
FPLM-1-r4	0.0288	3.2×10^{-5}	0.0288	0.83
FPLM-2-r4	0.0368	0.0416	0.0373	1.09
CLM-r4	0.0384	0.0833	0.0381	1.24
LAM [3]	0.0384	0.0833	0.0381	1.24
Half-Precision				
	MRED	|AE|	MRED	|AE|($\times 10^{-5}$)
FPLM-1	**0.0289**	**0.0021**	**0.0310**	**0.81**
FPLM-2	0.0365	0.0399	0.0394	1.10
FPLM-1-r4	0.0290	0.0043	0.0311	0.82
FPLM-2-r4	0.0362	0.0382	0.0392	1.10
CLM-r4	0.0397	0.0862	0.0416	1.27
LAM [3]	0.0391	0.0847	0.0409	1.26
Bfloat16				
	MRED	|AE|	MRED	|AE|($\times 10^{-5}$)
FPLM-1	**0.0302**	0.0175	**0.0300**	**0.73**
FPLM-2	0.0348	0.0280	0.0361	1.16
FPLM-1-r4	0.0330	0.0351	0.0326	0.74
FPLM-2-r4	0.0341	**0.0143**	0.0359	0.98
CLM-r4	0.0488	0.1066	0.0485	1.45
LAM [3]	0.0436	0.0950	0.0433	1.32
FP8				
	MRED	|AE|	MRED	|AE|($\times 10^{-5}$)
FPLM-1	0.2311	0.5626	0.2159	3.54
FPLM-2	**0.1626**	**0.3750**	**0.1586**	**2.61**
FPLM-1-r4	0.4367	1.0000	0.4232	8.69
FPLM-2-r4	0.3201	0.7500	0.3078	5.74
CLM-r4	0.3201	0.7500	0.3078	5.74
LAM [3]	0.1914	0.4375	0.1891	4.83

The MREDs for widths decreasing from 10 bits to 2 bits are presented due to the small accuracy drop from 23 bits to 11 bits. The bars and the lines with the marker show the MREDs for the FPM and all FP LMs, respectively. For the FPM, by considering single-precision as the baseline format, the reduction of the bit-width leads to accuracy loss in the product. The FP LMs only introduce errors in the mantissa computation and still computes an accurate exponent of the product. Therefore, we investigated the relation between the mantissa width and the error behavior with the same exponent width as in the single-precision, i.e., 8 bits. As shown in Fig. 5.1, the line with the circle marker shows the MREDs produced by the FPM for the uniform distribution. The lines with the triangle marker show the MREDs for the five proposed FPLMs and the LAM with respect to the uniform distribution. The FPM produces insignificant errors when the mantissa width is reduced from 10 bits to 7 bits, whereas its MRED greatly increases when the mantissa width is smaller than 7 bits. The MRED rapidly increases for all the LMs with decreases in the mantissa width from 6 bits to 2 bits. The FPLM-1-r4 has the largest error increase and the FPLM-2 becomes the most accurate, which produces smaller errors than the FPM when the mantissa width is lower than 5 bits.

The average multiplication results of 10^4 uniformly distributed random numbers within $[1, 2)$ are plotted with respect to the reduced mantissa widths, as shown in Fig. 5.2 and Fig. 5.3 for 7 bits and 4 bits to 2 bits due to the limit in space. To clarify the trends, the average values of products for each of the 50 samples were computed after arranging the products in the ascending order. As shown in 5.2 with decreasing mantissa width, the FPM generates increasingly smaller products (see the blue dots) compared to the single-precision baseline results (see the black dots). Meanwhile, since the LAM always results in smaller products (see the cyan plus sign) than the FPM, the error distances between the LAM and the baseline result will increase with a decreased mantissa widths. However, the FPLM-1 (see the green circles) and the FPLM-2 (see the orange squares) can produce both underestimated

and overestimated products. When the mantissa width is lower than 7-bit, the overestimated products have smaller error distances to the baseline products compared to the underestimated products, which eventually reduces the overall error distances. This trend is prominent especially when the mantissa width is from 4-bit to 2-bit, as shown in Fig. 5.2(d) to Fig. 5.2(f), where the overestimated products are closer to the baseline products. The FPLM-2 is more accurate when the mantissa width is from 5-bit to 2-bit since it produces a larger number of larger overestimated products than FPLM-1. FPLM-1-r4, FPLM-2-r4, and CLM-r4 show similar error behavior and smaller products compared to FPLM-1, FPLM-2, and LAM, respectively, whereas smaller products lead to larger error distances to the baseline products. Particularly, the error for FPLM-1-r4 rapidly rises when the mantissa width is 4-bit and becomes larger than CLM-r4 from 3-bit to 2-bit due to the increasing error distances for the larger inputs. When the mantissa width is larger than 6-bit, the FPLM-2-r4 is more accurate than the FPLM-2 since it computes a larger number of smaller errors for overestimated products. However, when the mantissa width is smaller than 7-bit, the error caused by the width truncation becomes dominant, leading to a larger number of underestimated products compared to the baseline results.

When the mantissa width is only 2-bit, the error behavior of the proposed multipliers is different from the LAM due to the additional error caused by the limited mantissa width. The truncation of the LSB for the FPLM-1 and the FPLM-2 introduces a relatively large error during the logarithm approximation, as shown in Fig. 5.2(f). When the mantissa is 1.75, its logarithm approximation is -0.25 and 0.75 for FPLM-1 and FPLM-2, respectively, which are the same as their approximate logarithm for the mantissa of 1.5. As a result, the multiplications for the mantissa of 1.75 produce the same results as that for the mantissa of 1.5. Due to the anti-logarithm approximation, the FPLM-1 produces more products that have the same value as the FPLM-2, as shown in Fig. 5.2(f). This leads to a significant error increase for FPLM-1 when the mantissa width is 2-bit. As shown in Fig. 5.3(f), the FPLM-1-r4

and FPLM-2-r4 have similar error behaviors, but with larger error distances. Due to many same products computed by the FPLM-1-r4, its underestimated errors are large when the mantissa width is 2-bit. The FPLM-2 and the CLM-r4 compute the same products due to the extremely small mantissa width.

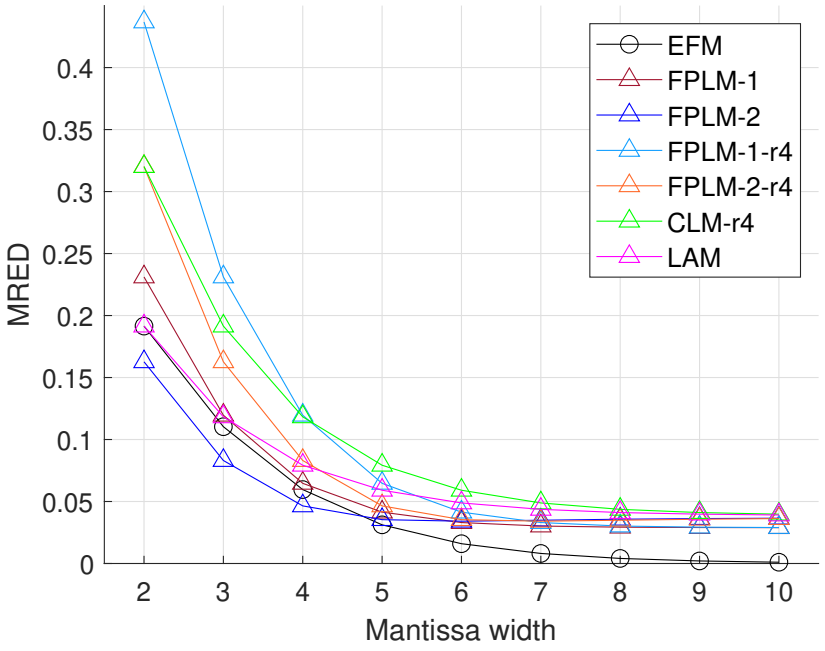
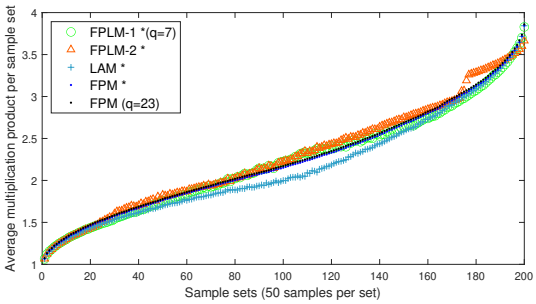
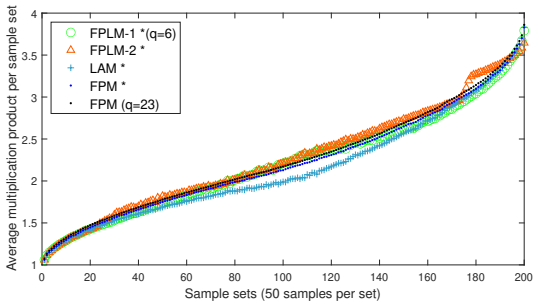


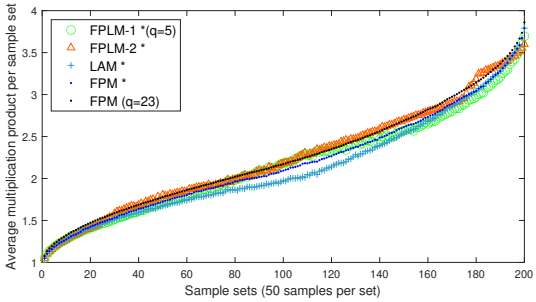
Figure 5.1: MREDs of the multipliers as a function of the mantissa width.



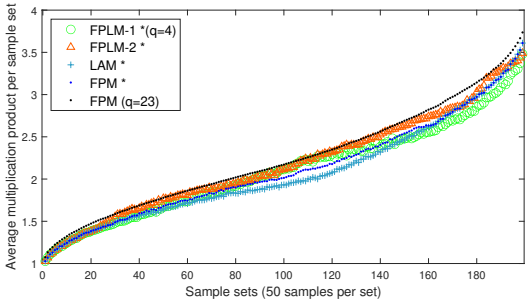
(a) For 7-bit mantissa.



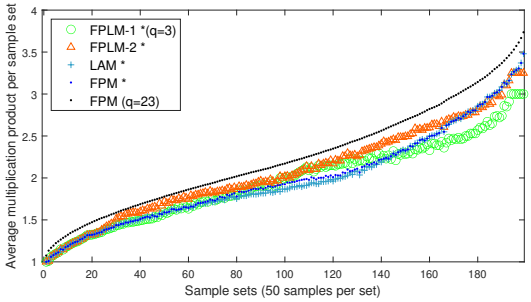
(b) For 6-bit mantissa.



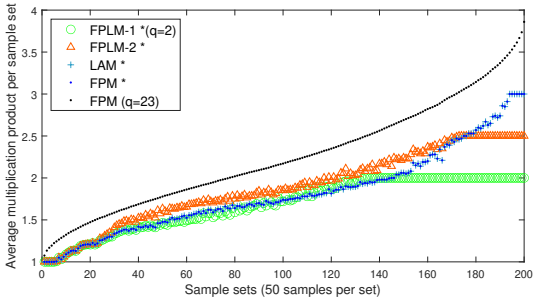
(c) For 5-bit mantissa.



(d) For 4-bit mantissa.

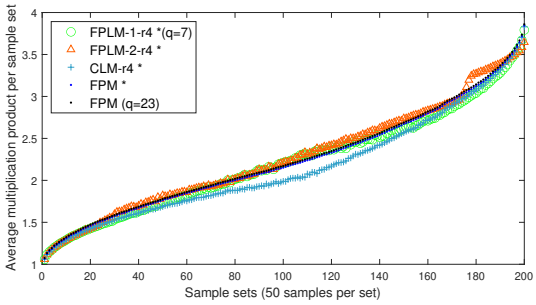


(e) For 3-bit mantissa.

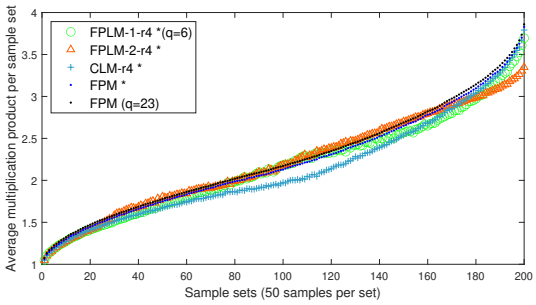


(f) For 2-bit mantissa.

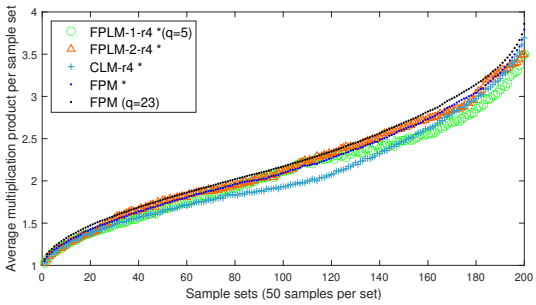
Figure 5.2: Average product of each sample set (50 samples per set) for FPLM-1, FPLM-2, LAM [3] and FPM with reduced mantissa width (from 7-bit to 2-bit) compared to the single-precision FPM (mantissa width of 23-bit).



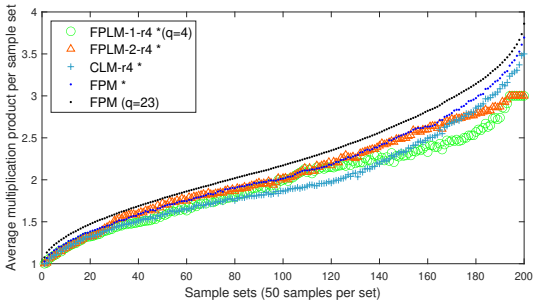
(a) For 7-bit mantissa.



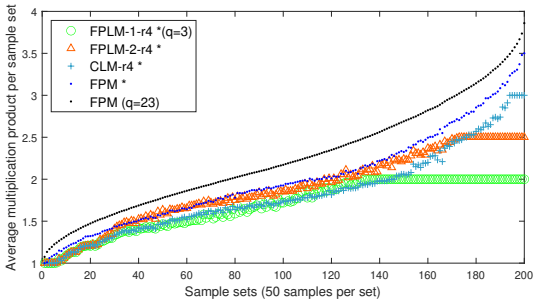
(b) For 6-bit mantissa.



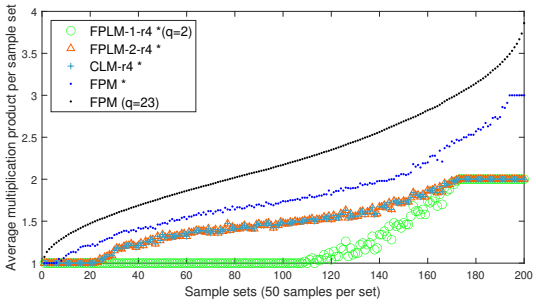
(c) For 5-bit mantissa.



(d) For 4-bit mantissa.



(e) For 3-bit mantissa.



(f) For 2-bit mantissa.

Figure 5.3: Average product of each sample set (50 samples per set) for FPLM-1-r4, FPLM-2-r4, CLM-r4 and FPM with reduced mantissa width (from 7-bit to 2-bit) compared to the single-precision FPM (mantissa width of 23-bit).

5.2 Hardware Evaluation

The five proposed FPLMs and the LAM in [3] were implemented in Verilog and an FPM was obtained using the Synopsys DesignWare IP library (DW_fp_mult). All of the designs were synthesized using the Synopsys Design Compiler (DC) for STM's CMOS 28-nm process with a supply voltage of $1.0\,V$ and a die temperature of $25°C$. All of the designs in the four precision formats were evaluated at a clock frequency of 250-MHz.

As shown in Table 5.2, for the 32-bit and 16-bit implementations, the proposed CLM-r4 is the most energy-efficient and smallest design. It incurs a smaller power-delay product (PDP) by 68× and a smaller area by 18× compared to the FPM for the single-precision implementation. The other four proposed designs have a larger hardware cost compared to the LAM while being more accurate. For the FP8 format, FPLM-1-r4 achieves a 19% less PDP and a 6.8% smaller area compared to LAM. The FPLM-2 consumes slightly smaller power and area compared to LAM while being 16% more accurate. FPLM-2-r4 and CLM-r4 are also more energy-efficient and smaller than LAM. It is important to note that, according to [34], the power of the accurate FP multiplier is dominated by the mantissa multiplication, accounting for over 80%, and the rounding unit for nearly 18%. Therefore, the reduction in power and area can be largely attributed to the elimination of the mantissa multiplier and the rounding unit in the proposed designs.

Table 5.2: Circuit Measurements of the FP Multipliers

	Power (μW)	Area (μm^2)	Delay (ns)	PDP (fJ)
Single-Precision				
FPM	643.4	2666	3.54	2277.6
FPLM-1	30.8	240.5	2.36	72.8
FPLM-2	25.8	211.9	2.20	56.9
FPLM-1-r4	29.9	234.1	2.27	67.9
FPLM-2-r4	22.8	198.6	2.11	48.2
CLM-r4	**17.3**	**146.7**	**1.92**	**33.2**
LAM	17.7	149.3	1.98	35.0
Half-Precision				
FPM	157.5	868.8	3.03	477.2
FPLM-1	15.1	119.9	1.25	18.9
FPLM-2	13.4	108.3	1.05	14.1
FPLM-1-r4	14.2	113.4	1.18	16.7
FPLM-2-r4	12.7	103.1	0.98	12.4
CLM-r4	**9.5**	**78.6**	**0.91**	**8.7**
LAM	9.9	81.2	0.97	9.6
Bfloat16				
FPM	107.8	724.6	2.90	312.6
FPLM-1	15.0	123.3	1.23	18.5
FPLM-2	14.1	115.7	1.05	14.8
FPLM-1-r4	14.1	116.8	1.16	16.3
FPLM-2-r4	13.4	110.4	0.98	13.1
CLM-r4	**11.2**	**93.8**	**0.91**	**10.2**
LAM	11.6	96.4	0.97	11.2
FP8				
FPM	41.0	327.5	2.10	86.14
FPLM-1	7.25	55.1	0.49	3.55
FPLM-2	7.14	54.1	0.48	3.42
FPLM-1-r4	**6.73**	**50.9**	**0.40**	**2.69**
FPLM-2-r4	6.81	52.0	0.40	2.72
CLM-r4	6.81	52.0	0.40	2.72
LAM	7.23	54.6	0.46	3.32

Chapter 6

Application Evaluations

6.1 JPEG Compression

6.1.1 Experimental Setup

The five proposed FP LMs are evaluated in JPEG compression for the four precision levels. The required matrix multiplications in the DCT and IDCT are implemented using the FP LMs. The quality of the compressed image is assessed by comparing it with the original image using the peak signal to noise ratio (PSNR).

The PSNR is based on the mean-square error (MSE), which represents the cumulative error between the compressed and the original image, as given by:

$$PSNR = 10\log_{10}(\frac{R^2}{MSE}), \tag{6.1}$$

where R is the maximum fluctuation in the input image data. A higher PSNR value indicates a higher image quality.

The JPEG compression of a 256×256-pixel "Lena" image is performed with the standard quantization matrix, Q, as given by:

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \tag{3}$$

The compression quality factor is set to 50 in the experiment.

6.1.2 Evaluation Results

The reconstructed images using different FP multipliers for the four precision levels are shown in Table 6.1 and the PSNRs are reported in Table 6.2.

As can be seen in Table 6.1, for 32-bit and 16-bit precision formats, images processed by FPLM-1, FPLM-2, FPLM-1-r4, and FPLM-2-r4 show similar quality as the accurate result, while images compressed using the LAM and CLM-r4 show a significant loss of quality. This indicates that the double-sided error distributions used in FPLM-1, FPLM-2, FPLM-1-r4, and FPLM-2-r4 can notably reduce the error accumulation in the multiple matrix multiplications in the DCT and IDCT. For the FP8 format, all reconstructed images show considerably lower quality due to the large errors produced by all of the FP multipliers. The PSNRs shown in Table 6.2 also confirm these results. Specifically, among the FP LMs, FPLM-1 achieves the highest PSNR for the single- and half-precisions, followed by FPLM-1-r4; FPLM-2 performs the best for the bfloat16 and FP8 formats, followed by FPLM-1. For 32-bit and 16-bit precisions, images processed by FPLM-1, FPLM-2, FPLM-1-r4, and FPLM-2-r4 show similar qualities as the accurate result, while images compressed using the LAM and CLM-r4 show significant losses in quality. This suggests that the double-sided error distributions produced by FPLM-1, FPLM-2, FPLM-1-r4, and FPLM-2-r4 can notably reduce error accumulation in the multiple matrix multiplications of the DCT

and IDCT. For the FP8 format, all reconstructed images show considerably lower quality due to the large errors produced by using FP LMs.

Overall, compared with LAM, the proposed FPLM-1, FPLM-2, FPLM-1-r4, and FPLM-2-r4 designs produce higher image qualities with a larger PSNR by up to 4.7dB; CLM-r4 offers higher energy and area savings with a very similar image quality. It is also shown that the obtained PSNRs follow the error analysis results for the FP LMs.

6.2 Neural Network Applications

6.2.1 Experimental Setup

The FP LMs are used in the training phase of a multi-layer perceptron (MLP) to illustrate their performance in NNs with respect to the classification accuracy and hardware performance. They are evaluated against the FPM considered as the baseline arithmetic unit, and the LAM [3]. In the experiments, the exact multiplication is replaced with approximate designs in both the training and inference phase by using the Pytorch framework [43]. The same multiplier is used both in the training phase and in the inference engine. Specifically, for each neuron, the approximate design is used to compute its output in the forward propagation, and the gradients of its weight and input in the backward propagation; the gradient of its output is computed by using exact multiplier, so the partial derivative operation is not modified in this case. It is noted that the training process is affected by many factors, which are explored for determining the training procedure for our experiments as discussed below.

Datasets and Network Models

Three classification datasets, fourclass [44], HARS [45] and MNIST [46], were used for the evaluation. A small MLP network was used in training for the fourclass dataset. The MLP networks used for the HARS and MNIST are (561, 40, 6) and (784, 128, 10) models, respectively. The activation function for the hidden layer is the Rectified

Table 6.1: JPEG compression using FP multipliers for the four precisions.

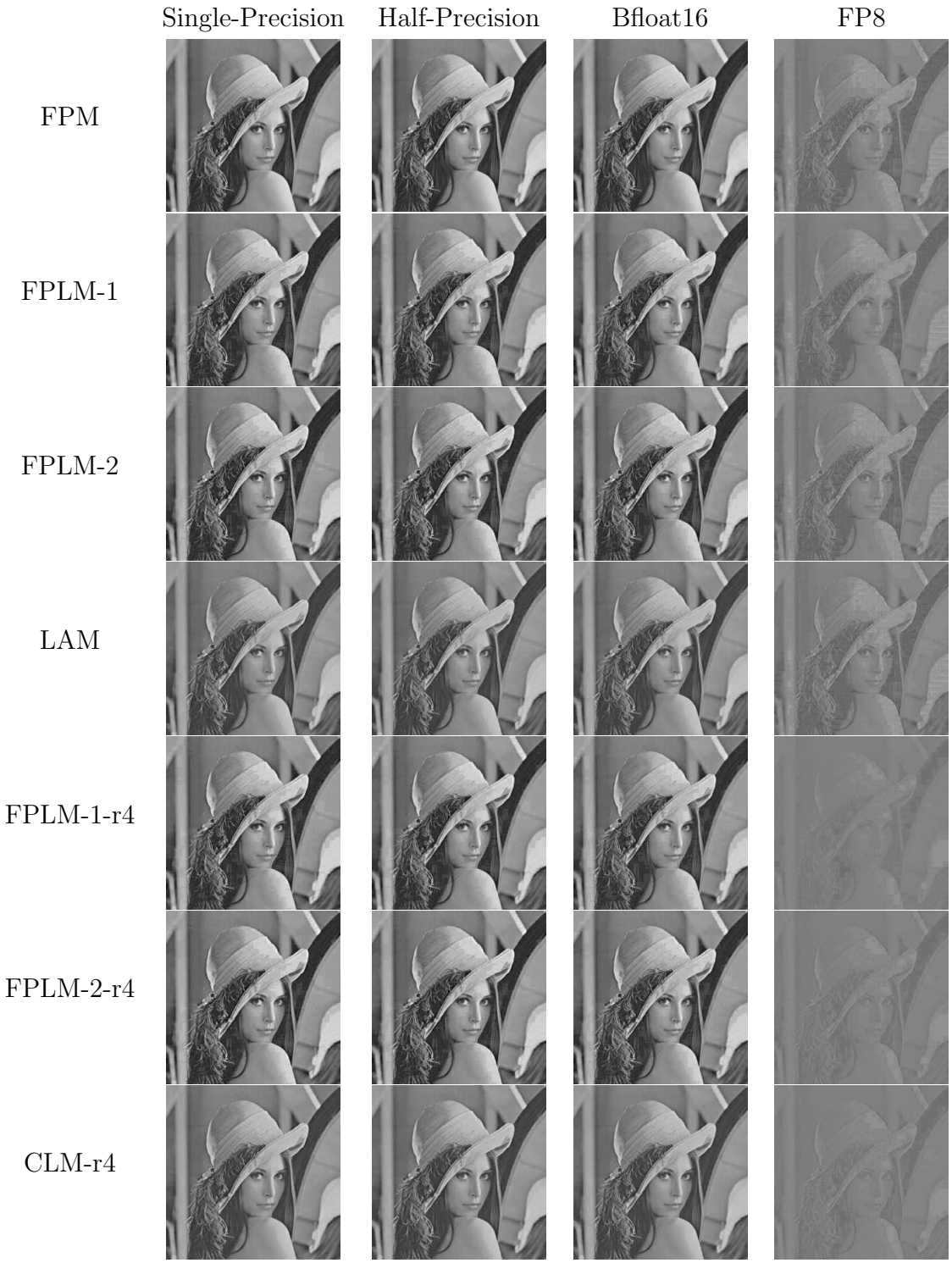


Table 6.2: PSNR (dB) for JPEG Compression using FP LMs with Different Precisions

	Single-Precision	Half-Precision	Bfloat16	FP8
FPM	33.92	33.91	33.71	17.43
FPLM-1	**31.91**	**31.90**	**31.56**	17.39
FPLM-2	31.86	31.87	31.44	**17.48**
FPLM-1-r4	31.91	31.90	31.15	15.50
FPLM-2-r4	31.86	31.87	30.99	15.77
CLM-r4	27.22	27.16	26.17	15.77
LAM	27.22	27.17	26.60	17.43

Linear Unit (ReLU) function; for the output layer is the sigmoid function and the softmax function for the fourclass and the other two datasets, respectively. The ReLU is considered here since it is simple to implement and mitigates the vanishing gradients problem [47]. The cross entropy loss function is adopted for calculating the loss [48].

Hyperparameter Configurations

To fairly evaluate the effect of approximate multiplication on training, for each dataset the same training procedure is used for all the models using different FP multipliers. The optimizer, i.e., stochastic gradient descent [41], with default parameters and the initialization method that is default in the Pytorch framework, is utilized with no learning rate decay, the same splits for train/validation/test set sizes, the same maximum number of training epochs, and the same early stopping criterion. An NN was trained by employing approximate multipliers with four FP precisions and using the same training procedure. In all cases, the mini-batch training strategy [40] is used; the batch size for the MNIST, HARS and fourclass are set to 128, 300 and 100, respectively. In addition, five trials were done using different random initialization for each training simulation.

Early Stopping Criterion

The stopping criterion has an important impact on obtaining the trained parameters, i.e., weights and biases, and thereby affects the classification accuracy. The selection of the early stopping criterion is usually in an ad-hoc fashion and significantly involves the consideration of a trade-off between the training time and generalization error [49]. In order to obtain optimal generalization performance and to avoid overfitting, an early stopping strategy terminates the training if the validation loss does not improve for a number of (t) epochs or the set maximum number of epochs is reached. The trained parameters is then obtained at the epoch that achieves the lowest validation loss. The maximum number of training epochs for the three datasets are set to 30k epochs; at each epoch, the entire training set is shuffled. Note that the actual training time is much shorter due to the early stopping criterion. We exploited this early stopping criterion for adapting to each dataset and each precision format since the validation loss can be different in various cases with respect to the number of epochs and the local minima before reaching the global minimum [49]. We observed in our experiment that, given $t < 100$, the early stopping criterion is likely to be satisfied for the three datasets whereas the training is stuck in a local minimum. Among the three datasets, it is also observed that the dataset of a smaller size is easier to get stuck. Therefore, based on our observation, t is set to 100, 200, and 1000 for MNIST, HARS, and fourclass, respectively, which helps achieve an acceptable trade-off between training time and validation loss.

6.2.2 Evaluation Results

Classification Accuracy Analysis

The benchmark NNs that use FPMs for training are considered as baseline models. The comparison of the average classification accuracy of five trials is shown in Table 6.3 with the baseline NN classification accuracies for the four precisions to indicate the relative approximation error of the FP LMs. The result of each trial is shown in

Fig. 6.1, Fig. 6.2, and Fig. 6.3 for the MNIST, HARS and Fourclass, respectively.

No FP LM dominates across all the datasets and the precision formats. For each trial, the classification accuracy fluctuates for each specific multiplier for different datasets, FP precision formats, and random initialization. However, the differences are subtle, especially for 32-bit and 16-bit precisions. As shown in Fig. 6.1(a)-(c) and Fig. 6.2(a)-(c) For each trial, the difference in classification accuracy is $< 1\%$ in most cases for these multipliers. Considering the average results, as shown in Table 6.3, the difference is even smaller ($< 0.4\%$) for MNIST and HARS. This average difference is slightly larger for fourclass ($< 2\%$) in the 32-bit and 16-bit precision formats. For the FP8 format, the FPLM-1-r4 degrades the most across all the datasets due to its large error, which also leads to the largest accuracy difference of $2\% - 4\%$. Among other multipliers, the difference is smaller than 2% for all the datasets. Overall, the evaluation result indicates that NNs using the five proposed multipliers and LAM can lead to very close classification accuracies for MNIST and HARS with the 32-bit and 16-bit precision formats, while the accuracies of classifying a small dataset, i.e., fourclass, or using the FP8 format have slightly larger differences.

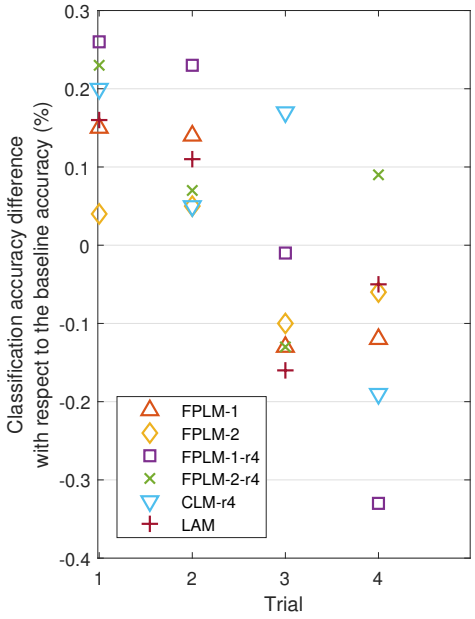
For the MNIST dataset, CLM-r4 achieves the highest average classification accuracy in the single-precision and bfloat16 formats, followed by FPLM-2-r4 and FPLM-2, respectively. The FPLM-2-r4 performs the best in the half-precision format, followed by the FPLM-1. It is interesting to observe that all of the multipliers slightly improve the classification accuracy for the single-precision format. For the HARS, the FPLM-1 performs the best at single- and half-precision, while CLM-r4 results in the best accuracy in the bfloat16 format. For fourclass, the FPLM-1 is the most accurate in the single-precision and the bfloat16 formats, while CLM-r4 and LAM achieves the same best accuracy in the half-precision format. For the FP8 format, the FPLM-2 outperforms the other multipliers for both MNIST and fourclass while it improves the classification accuracy for all the datasets; the FPLM-1 achieves the best for HARS with an improved accuracy.

The classification accuracy is not strictly consistent with the error analysis results for the FP LMs. It indicates that a smaller multiplication error does not always lead to a higher classification accuracy. However, for the FP8 format, the performance of using different LMs loosely follows the error analysis results for the multipliers in Table 5.1. This is possible since the iterative computation in the training process including forward and backward propagations depends on many different factors. These factors vary for each dataset, network structure and random initialization.

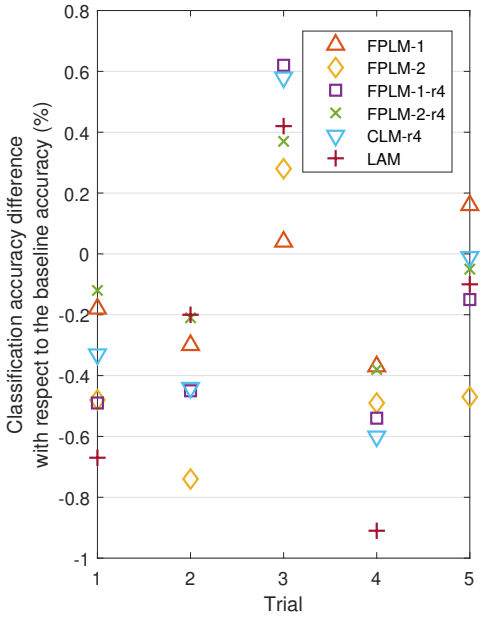
Hardware Evaluation

An artificial neuron was implemented to assess the overall hardware cost of NNs. The FP adder used in the neuron was obtained using the Synopsys DesignWare IP library (DW_fp_add).

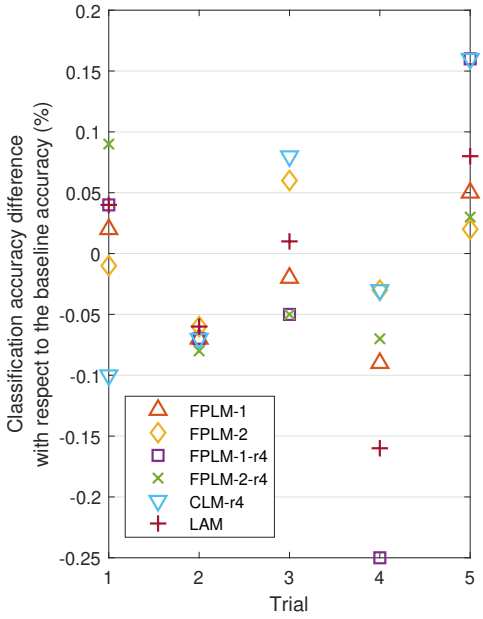
The simulation results in Table 6.4 are obtained at a clock frequency of 125 MHz. It is shown that a neuron using the CLM-r4 dissipates the least energy with the smallest area for the 32-bit and 16-bit precisions, which is up to 6.1× smaller in energy with an area up to 6.2× smaller than the neuron using FPMs. For the FP8 format, the FPLM-1-r4 is the most energy-efficient and the smallest design while the FPLM-2, FPLM-2-r4, and CLM-r4 all perform better than the LAM. In general, the hardware improvements are smaller than that in Table 5.2 since they are limited by the FP adder, which has larger hardware overhead.



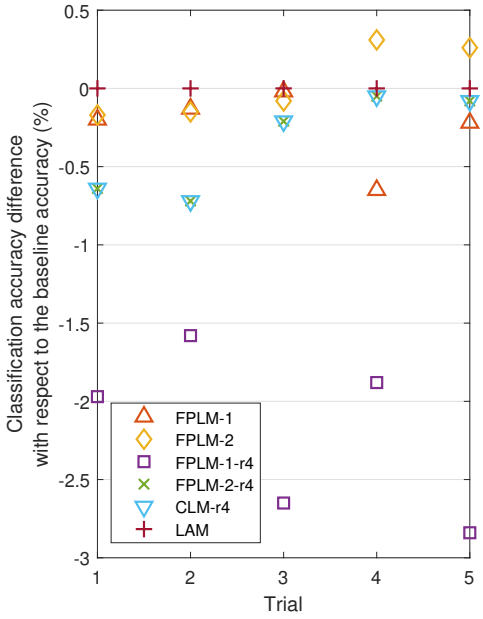
(a) For single-precision.



(b) For half-precision.

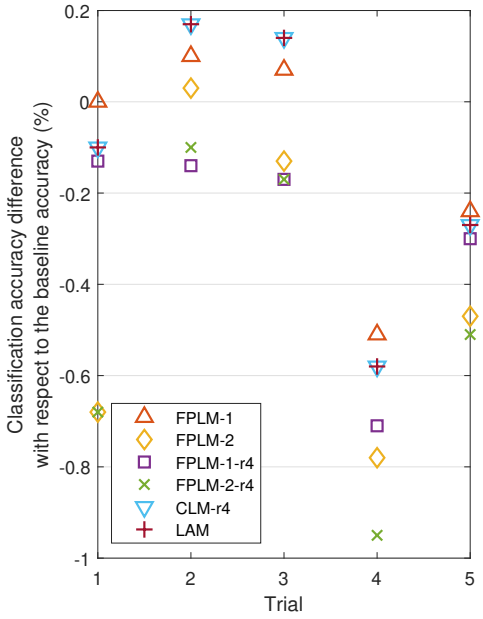


(c) For bfloat16 format.

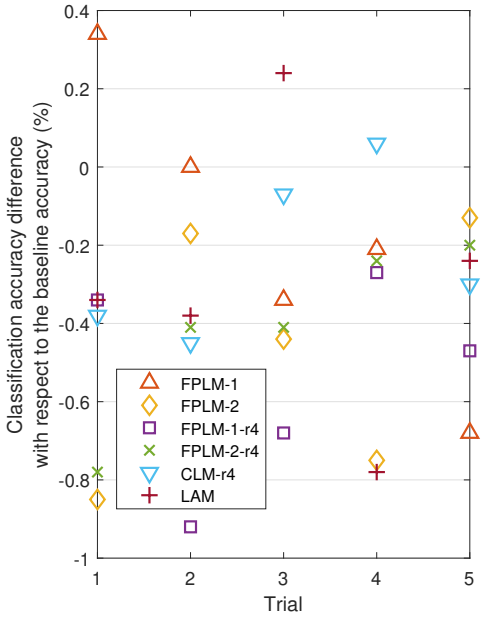


(d) For FP8 format.

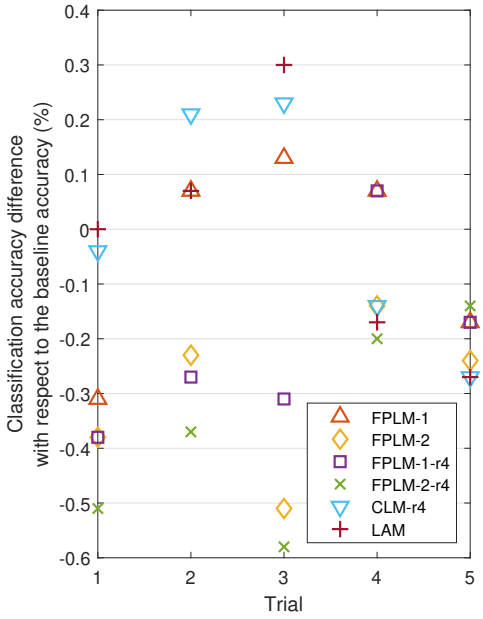
Figure 6.1: Comparison of the classification accuracy of the MNIST dataset with LMs for four precision levels: a negative percentage means a decrease and a positive percentage means an increase in the accuracy with respect to using accurate multipliers.



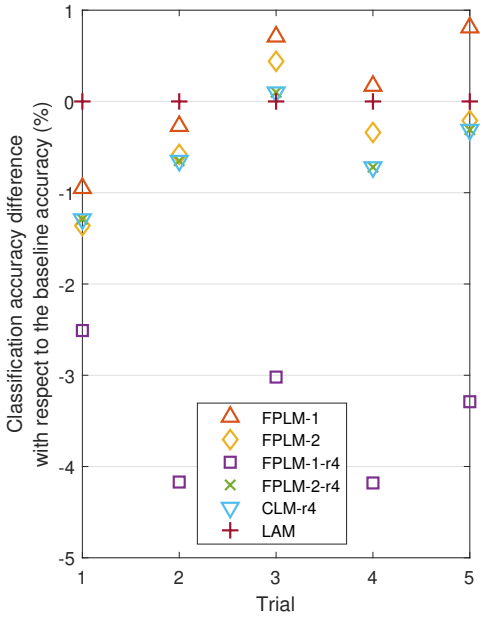
(a) For single-precision.



(b) For half-precision.

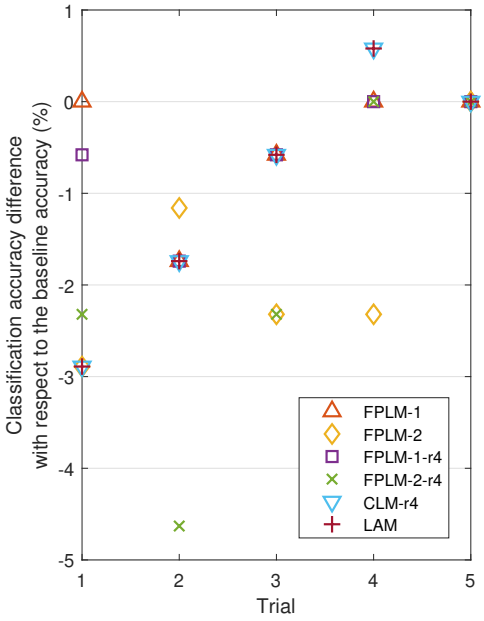


(c) For bfloat16 format.

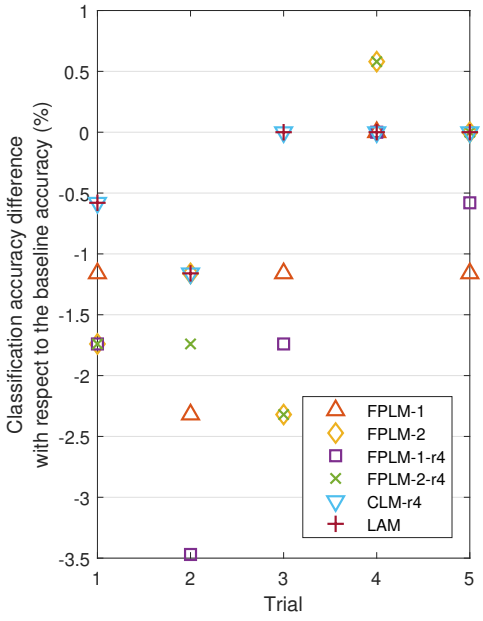


(d) For FP8 format.

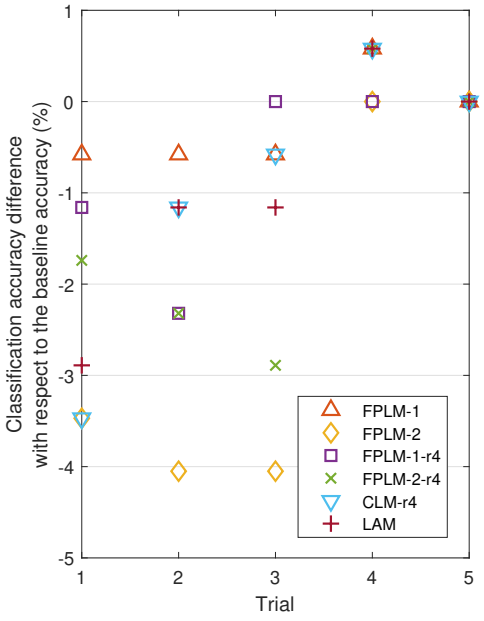
Figure 6.2: Comparison of the classification accuracy of the HARS dataset with LMs for four precision levels: a negative percentage means a decrease and a positive percentage means an increase in the accuracy with respect to using accurate multipliers.



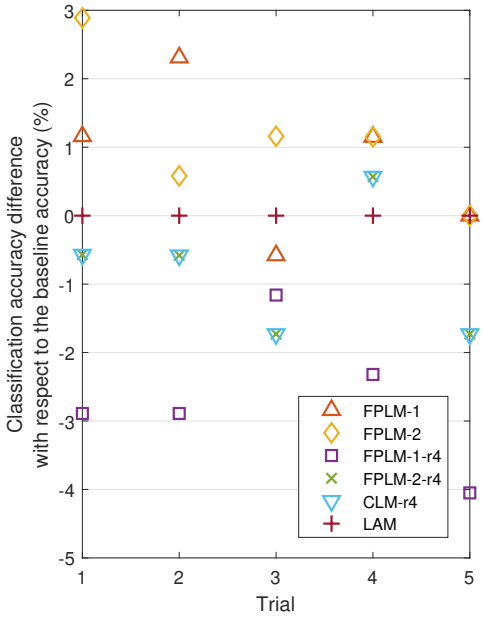
(a) For single-precision.



(b) For half-precision.



(c) For bfloat16 format.



(d) For FP8 format.

Figure 6.3: Comparison of the classification accuracy of the Fourclass dataset with LMs for four precision levels: a negative percentage means a decrease and a positive percentage means an increase in the accuracy with respect to using accurate multipliers.

Table 6.3: Average classification accuracy of three datasets with LMs for four precision levels

	Single-Precision	Half-Precision	Bfloat16	FP8
MNIST				
FPM	97.85	96.75	97.91	96.52
FPLM-1	97.88	96.62	97.88	96.28
FPLM-2	97.89	96.37	97.90	**96.55**
LAM	97.87	96.46	97.89	96.52
FPLM-1-r4	97.89	96.55	97.87	94.34
FPLM-2-r4	97.91	**96.68**	97.89	96.18
CLM-r4	**97.93**	96.59	**97.91**	96.18
HARS				
FPM	96.07	95.52	96.01	94.38
FPLM-1	**95.96**	**95.34**	95.96	**94.47**
FPLM-2	95.67	95.05	95.71	93.97
LAM	95.94	95.22	95.99	94.38
FPLM-1-r4	95.78	94.98	95.79	90.94
FPLM-2-r4	95.59	95.11	95.65	93.80
CLM-r4	95.94	95.29	**96.00**	93.80
Fourclass				
FPM	99.42	99.54	99.42	92.48
FPLM-1	**98.96**	98.38	**99.19**	93.29
FPLM-2	97.68	98.61	97.11	**93.64**
LAM	98.49	**99.19**	98.49	92.48
FPLM-1-r4	98.84	98.03	98.72	89.82
FPLM-2-r4	97.57	98.49	98.15	91.67
CLM-r4	98.49	99.19	98.49	91.67

Table 6.4: Circuit Assessment of the Artificial Neuron

	Power(μW)	Area(μm^2)	Delay(ns)	PDP(fJ)
Single-Precision				
FPM	773.3	6362.35	7.70	5954.41
FPLM-1	155.6	1200.49	7.63	1187.22
FPLM-2	149.4	1148.27	7.43	1110.04
FPLM-1-r4	153.3	1180.58	7.58	1162.01
FPLM-2-r4	147.1	1127.71	7.29	1072.35
CLM-r4	**134.8**	**1022.93**	**7.17**	**966.51**
LAM	135.3	1025.87	7.24	979.57
Half-Precision				
FPM	207.5	2025.14	6.34	1315.55
FPLM-1	75.97	554.39	4.58	347.94
FPLM-2	74.41	533.5	4.37	325.17
FPLM-1-r4	73.29	536.27	4.46	326.87
FPLM-2-r4	71.93	517.67	4.26	306.42
CLM-r4	**67.68**	**472.3**	**4.17**	**282.22**
LAM	69.49	485.52	4.28	297.41
Bfloat16				
FPM	144.2	1075.11	6.53	941.62
FPLM-1	59.18	537.09	4.45	263.35
FPLM-2	57.93	522.4	4.27	247.36
FPLM-1-r4	57.69	526.32	4.38	252.68
FPLM-2-r4	56.93	514.89	4.21	239.67
CLM-r4	**54.17**	**483.23**	**4.13**	**223.72**
LAM	54.83	486.17	4.20	230.28
FP8				
FPM	56.64	760.18	4.48	253.74
FPLM-1	29.64	256.87	2.91	86.25
FPLM-2	28.48	251.65	2.87	81.73
FPLM-1-r4	**25.03**	**221.29**	**2.71**	**67.83**
FPLM-2-r4	27.79	240.55	2.79	77.53
CLM-r4	27.79	240.55	2.79	77.53
LAM	29.97	256.55	2.86	85.71

Chapter 7

Conclusions

This thesis starts with an introduction to the growing challenge for computation-intensive applications and the motivation that uses approximate computing techniques for designing hardware-efficient multipliers. Approximate multiplier designs are reviewed with a discussion of computationally-demanding arithmetic operations in image processing and NN applications in Chapter 2.

In Chapter 3, two novel approximation methods are proposed for the logarithm and anti-logarithm conversions. Double-sided error distributions are produced in both approximation methods, which helps to reduce the error accumulation in MAC operations. A detailed analysis of the approximation error is presented for the proposed two approximation methods, which shows an analytical scheme for examining errors in logarithm approximation.

The circuit designs for the proposed FP logarithmic multipliers are discussed in Chapter 4. The circuits are optimized for the approximation methods and the radix-4 logarithm is used to further reduce the hardware complexity. In Chapter 5, the performance evaluations show that the five proposed multipliers provide different trade-offs between accuracy and hardware performance. For the 32-bit and 16-bit precisions, the proposed FPLM-1 multiplier is the most accurate design with up to 30.8% smaller MRED and $10^3\times$ smaller average error compared to LAM. CLM-r4 is the most energy-efficient multiplier and the smallest design with up to 68× smaller

PDP and up to 18× smaller area compared to the conventional FP multiplier. For the FP8 format, FPLM-1-r4 is the most energy-efficient and the smallest design. From the experimental results, the double-sided error distribution benefits from error cancellation in low-precision computations.

Finally, the use of the proposed FP logarithmic multipliers in image processing and NN applications is explored in Chapter 6. Using the proposed FP logarithmic multipliers in JPEG image compression achieves higher image quality than LAM, with a larger PSNR by up to 4.7dB. Compared to NNs implemented using a conventional FP multiplier, the evaluation for several benchmark NNs shows up to 6.1× and 6.2× reduction in energy consumption and area cost, respectively, while achieving very close classification accuracies. Moreover, higher classification accuracies can be obtained by using the proposed designs compared to the use of the conventional FP multiplier. We also found that no single FP logarithmic multiplier performed the best across all the datasets with different precisions. Therefore, designing FP logarithmic multipliers from the application's perspective could be more efficient. To select an appropriate FP logarithmic multiplier for an NN application, the impact of FP logarithmic multipliers on the training process of NNs remains an important topic for future research.

Bibliography

[1] P. Yin, C. Wang, W. Liu, E. E. Swartzlander, and F. Lombardi, "Designs of approximate floating-point multipliers with variable accuracy for error-tolerant applications," in *J. Signal Process. Syst.*, vol. 90, 2018, pp. 641–654.

[2] D. Peroni, M. Imani, and T. S. Rosing, "Runtime efficiency-accuracy tradeoff using configurable floating point multiplier," in *IEEE Trans. Comput.-Aided Des. of Integr. Circuits and Syst.*, vol. 39, 2020, pp. 346–358.

[3] T. Cheng, Y. Masuda, J. Chen, J. Yu, and M. Hashimoto, "Logarithm-approximate floating-point multiplier is applicable to power-efficient neural network training," in *Integration*, vol. 74, 2020, pp. 19–31.

[4] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, 2009, pp. 850–862.

[5] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," in *IEEE Trans. on Comput.-Aided Des. of Integr. Circuits and Syst.*, vol. 32, 2012, pp. 124–137.

[6] M. S. Ansari, B. F. Cockburn, and J. Han, "An improved logarithmic multiplier for energy-efficient neural computing," in *IEEE TC*, vol. 70, 2021, pp. 614–625.

[7] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," in *Proc. IEEE*, vol. 105, 2017, pp. 2295–2329.

[8] Q. Zhang, M. Zhang, T. Chen, Z. Sun, Y. Ma, and B. Yu, "Recent advances in convolutional neural network acceleration," in *Neurocomputing*, vol. 323, 2019, pp. 37–51.

[9] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *DAC*, 2013, pp. 1–9.

[10] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *ETS*, 2013, pp. 1–6.

[11] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," in *Proc. IEEE*, vol. 108, 2020, pp. 2108–2135.

[12] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. 32nd Int. Conf. on Machine Learning*, vol. 37, 2015, pp. 1737–1746.

[13] M. Yan, Y. Song, Y. Feng, G. Pasandi, M. Pedram, and S. Nazarian, "Knncam: A k-nearest neighbors-based configurable approximate floating point multiplier," in *ISQED*, 2019, pp. 1–7.

[14] M. Imani, R. Garcia, S. Gupta, and T. Rosing, "Rmac: Runtime configurable floating point multiplier for approximate computing," in *Proc. Int. Symposium on Low Power Electronics and Design*, 2018, pp. 1–6.

[15] V. Camus, J. Schlachter, C. Enz, M. Gautschi, and F. K. Gurkaynak, "Approximate 32-bit floating-point unit design with 53% power-area product reduction," in *ESSCIRC*, 2016, pp. 465–468.

[16] S. i. O'uchi, H. Fuketa, T. Ikegami, *et al.*, "Image-classifier deep convolutional neural network training by 9-bit dedicated hardware to realize validation accuracy and energy efficiency superior to the half precision floating point format," in *ISCAS*, 2018, pp. 1–5.

[17] M. Franceschi, A. Nannarelli, and M. Valle, "Tunable floating-point for artificial neural networks," in *ICECS*, 2018, pp. 289–292.

[18] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong, "Lognet: Energy-efficient neural networks using logarithmic computation," in *ICASSP*, 2017, pp. 5900–5904.

[19] X. Sun, N. Wang, C. Chen, *et al.*, "Ultra-low precision 4-bit training of deep neural networks," in *NeurIPS*, 2020, pp. 1796–1807.

[20] N. Wang, J. Choi, D. Brand, C. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *NeurIPS*, 2018, 7685–7694.

[21] J. N. Mitchell, "Computer multiplication and division using binary logarithms," in *IRE Trans. on Electronic Computers*, vol. 11, 1962, pp. 512–517.

[22] V. Mahalingam and N. Ranganathan, "Improving accuracy in Mitchell's logarithmic multiplication using operand decomposition," in *IEEE TC*, vol. 55, 2006, pp. 1523–1535.

[23] D. Nandan, J. Kanungo, and A. Mahajan, "An efficient VLSI architecture design for logarithmic multiplication by using the improved operand decomposition," in *Integration*, vol. 58, 2017, pp. 134–141.

[24] Z. Babić, A. Avramović, and P. Bulić, "An iterative logarithmic multiplier," in *Microprocessors and Microsystems*, vol. 35, 2011, pp. 23–33.

[25] S. E. Ahmed and M. Srinivas, "An improved logarithmic multiplier for media processing," in *J. Signal Process. Syst.*, 2019, pp. 561–574.

[26] H. Kim, M. S. Kim, A. A. D. Barrio, and N. Bagherzadeh, "A cost-efficient iterative truncated logarithmic multiplication for convolutional neural networks," in *ARITH*, 2019, pp. 108–111.

[27] M. S. Kim, A. A. D. Barrio, L. T. Oliveira, R. Hermida, and N. Bagherzadeh, "Efficient Mitchell's approximate log multipliers for convolutional neural networks," in *IEEE TC*, vol. 68, 2019, pp. 660–675.

[28] W. Liu, J. Xu, D. Wang, C. Wang, P. Montuschi, and F. Lombardi, "Design and evaluation of approximate logarithmic multipliers for low power error-tolerant applications," in *IEEE TCAS-I: Regular Papers*, vol. 65, 2018, pp. 2856–2868.

[29] R. Pilipović, P. Bulić, and U. Lotrič, "A two-stage operand trimming approximate logarithmic multiplier," in *IEEE TCAS-I: Regular Papers*, vol. 68, 2021, pp. 2535–2545.

[30] P. Yin, C. Wang, H. Waris, W. Liu, Y. Han, and F. Lombardi, "Design and analysis of energy-efficient dynamic range approximate logarithmic multipliers for machine learning," in *IEEE Trans. on Sustainable Computing*, vol. 6, 2020, pp. 612–625.

[31] J. Y. Tong, D. Nagle, and R. A. Rutenbar, "Reducing power by optimizing the necessary precision/range of floating-point arithmetic," in *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 8, 2000, pp. 273–286.

[32] F. Fang, T. Chen, and R. A. Rutenbar, "Floating-point bit-width optimization for low-power signal processing applications," in *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, vol. 3, 2002, pp. III–3208–III–3211.

[33] J. Eilert, A. Ehliar, and D. Liu, "Using low precision floating point numbers to reduce memory cost for MP3 decoding," in *IEEE 6th Workshop on Multimedia Signal Processing*, 2004, pp. 119–122.

[34] A. Gupta, S. Mandavalli, V. J. Mooney, *et al.*, "Low power probabilistic floating point multiplier design," in *ISVLSI*, 2011, pp. 182–187.

[35] H. Zhang, W. Zhang, and J. Lach, "A low-power accuracy-configurable floating point multiplier," in *ICCD*, 2014, pp. 48–54.

[36] G. K. Wallace, "The JPEG still picture compression standard," in *IEEE Trans. on consumer electronics*, vol. 38, 1992, pp. xviii–xxxiv.

[37] A. C. Bovik, *The essential guide to image processing*. Academic Press, 2009.

[38] J. Schmidhuber, "Deep learning in neural networks: An overview," vol. 61, 2015, pp. 85–117.

[39] S. Haykin, *Neural networks and learning machines, 3/E*. Pearson Education India, 2009.

[40] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient mini-batch training for stochastic optimization," in *Proc. of the 20th ACM SIGKDD international conf. on Knowledge discovery and data mining*, 2014, pp. 661–670.

[41] H. Robbins and S. Monro, "A stochastic approximation method," in *The annals of mathematical statistics*, 1951, pp. 400–407.

[42] "IEEE standard for binary floating-point arithmetic," 1985, pp. 1–20.

[43] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, 2019, 8026–8037.

[44] C. Chang and C. Lin, *Fourclass*, 1996. [Online]. Available: https://www.csie.ntu.edu.tw/cjlin/libs-vmtools/datasets/binary.html.

[45] U. M. L. Repository, *Human activity recognition using smartphones data set*, 2012. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones.

[46] Y. LeCun, C. Cortes, and C. Burges, *MNIST handwritten digit database*, 2001. [Online]. Available: http://yann.lecun.com/exdb/mnist.

[47] G. I, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[48] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[49] L. Prechelt, "Early stopping-but when?" In *Neural Networks: Tricks of the trade*, 1998, pp. 55–69.