Waseda University Doctoral Dissertation

Research on Approximate Multipliers Based on Probability-Driven Carry-Restricted Compressors for Error-Tolerant Applications

Yi GUO

Graduate School of Information, Production and Systems

Waseda University

April 2021

Abstract

Along with the emergence of more and more complex applications in computing systems, the overall computational workloads and energy consumption of the systems are continuously increasing. Lots of applications include huge number of computations and small accuracy loss can be accepted. Such applications are defined as error-tolerant applications. To realize the good product of applications, LSI implementation is necessary. Among implementation, optimized multiplier is important because it is as the basic unit of lots of applications. During the application design, accuracy and hardware resource need to be considered. Approximate computing has been a promising technique to balance the accuracy quality and hardware resource. 8-bit multiplications are utilized in applications for manipulating images such as filtering operations and classification using convolutional neural networks. Therefore, 8x8 approximate multiplier design has attracted lots of attentions. There are mainly two important kinds of approximate multiplier: ASIC-based approximate multiplier, which is main focus and the most intuitive methodology; and FPGA-based approximate multiplier, which is a popular choice as changes of applications due to the reconfigurability and fast development round of FPGA. Approximation certainly incurs the accuracy loss in the design, therefore the trade-off between accuracy loss and hardware saving is the key target of approximate multiplier.

Approximate multiplier is designed based on three steps of conventional exact multipliers, partial product generation, accumulation and final carry propagate adder. Firstly, a partial product matrix of 8 rows and 8 columns is generated by AND operation on two 8-bit input operands. Secondly, partial product matrix is accumulated and reduced to 2 rows by adding partial products at the same position using full adders or compressors. A full adder can add (compress) three partial products and generate 1-bit result (i.e. sum) at the same position and another bit result (i.e. carry) at 1-bit higher position. Thirdly, final carry propagate adder processes 2 rows and produces the multiplication result.

As for ASIC-based approximate multiplier, method of propagation restriction using only OR operation was discussed in Qiqieh's work [DATE, 2017] and Yadav's work [MWSCAS, 2018]. The method is simple to reduce the circuit complexity, but the error is high. Another method of propagation restriction was discussed in Boroumand's work [ASP-DAC, 2018] by dividing the carry to two inexact parts. The parallel operation of two semi-carries simplifies the computation, but this method is limited to 3-input compressor. The method of less-XOR compressors was discussed in three approximate multipliers of Yang's work [ICCD, 2017], Venkatachalam's work [TVLSI, 2017], and Jiang's work [TCAS-I, 2019]. All can simplify the compressor with less XOR gates, but the remaining XOR gates limits the potential of effective synthesis.

As for FPGA-based approximate multipliers, Look-Up-Table (LUT) based accumulation and special carry chain for the carry propagation are considered. One multiplier was introduced by Ullah [DAC, 2018], which used inexactly predicted carry-in as one input of approximate compressor. Thus, the computation of each position is independent with each other. The same author discussed another approximate multiplier in [DAC, 2018], by omitting one input in the compressor to save LUTs. Two approaches both can optimize the hardware utilization. However, they have a common issue that the approximation on input would affect both of two outputs of one compressor. It incurs the high accuracy loss.

To achieve trade-off between accuracy loss and hardware saving, and to solve the limitation of previous works, probability-driven carry-restricted compressors are proposed. Because the carry of compressor occurs in rare cases, the major approximation on carry can save the hardware and ensure the low error. As for ASIC-based multiplier, the compressor is implemented by logic gates. The conventional compressor usually generates the carry to higher position. By caring about the low 1-probability of one partial product, new compressors with large size (4 to 2, 8 to 2, etc.) have been developed to generate 2-bit outputs at the same position without generating a carry to higher position. ASIC circuit can be simplified by avoiding XOR gates which are most consuming gate among all logic gates. As for FPGA-based multiplier, the compressors are implemented with LUTs. Each carry of a compressor costs one

corresponding LUT. Because the carry probability of each position to high position is low, the carry computation is omitted in the proposed compressor. By doing so, the LUT resource can be saved. Moreover, three types of multipliers are proposed by using different number of proposed compressors.

This dissertation is organized as follows:

In Chapter 1 [Introduction], background of approximate computing and the necessity of approximate multiplier are introduced. Then, the research status on approximate multiplier and existing issues are discussed. Finally, the motivations, target and proposed concept are given.

In Chapter 2 [ASIC-Based Approximate Multiplier using Probability-Driven Inexact Compressors], an ASIC-based multiplier with inexact compressors and error recovery is proposed, which efficiently trades accuracy for hardware efficiency. To accumulate partial products with low-cost circuits, inexact compressors are proposed by analyzing the probability of partial product matrix and restricting carry generation. This compressor design can reduce eight rows into two rows with only one stage. In addition, an error-recovery scheme is proposed to compensate error. Different from the existing bit-wise error recovery, the proposed error-recovery scheme processes the error elements in the form of group. Thus, the critical path of multiplier can be shortened. As a result, in terms of mean relative error distance (MRED), the accuracy loss of the proposed multiplier is as low as 1.07%. Compared with the exact multiplier using 40nm process, the proposed multiplier can reduce power by 59.75% and area by 42.47%. The delay reduction is larger than 12.78%. Compared with the previous approximate multipliers, the proposed multiplier has a better accuracy-hardware result.

In Chapter 3 [FPGA-Based Approximate Multiplier using Carry-Inexact Elementary Modules], an 8x8 FPGA-based multiplier is proposed based on 4x4 carry-inexact multipliers and one inexact adder. In the proposed 4×4 multiplier, the compressor accumulating partial products is implemented by LUTs. Because the carry result of compressor occurs in rare situation, the carry computation is omitted in proposed design. Approximate 8×8 multiplier is built from four 4×4 multipliers with an adder. To fast produce the final product, two types of inexact adder are proposed, where

the result of each bit is produced in parallel and the critical path is shortened. In terms of MRED, the error of the proposed 8×8 multiplier is as low as 1.06%. Compared with the exact multiplier, the proposed design can reduce area by 43.66% and power by 24.24%. The delay saving is up to 29.50%. To comprehensively evaluate the performance of approximate multipliers, a Pareto-optimal analysis is discussed. The proposed design has more Pareto-optimal points, which means the proposed design has a better accuracy-hardware result than previous approximate multipliers.

In Chapter 4 [Conclusion and Future Work], the overall dissertation is summarized and the future works are presented. To realize approximate multiplier with a considerable accuracy-hardware trade-off, approach for ASIC-based and FPGA-based multiplier is proposed. In the future, large-size multiplier, floating-point multiplier will be extended.

Acknowledgement

First of all, I would like to show my great respect and thanks to my supervisor, Professor Shinji Kimura, for his guidance in my research work. Pursing a doctoral degree is a hard and long journey, which I obviously can't finish without his patient instruction. Professor Kimura talks with me about my research topic, which broadens my views and enlightens me for my research. Every time when I did the seminars or the communication with him, Professor Kimura always gives me meaningful advices. I feel so grateful and lucky that I could be one of the students of Professor Kimura. The past five years of studying in Professor Kimura's Lab. are the years I have harvested the most and grown the fastest.

Secondly, I would like to express my sincere thanks to Professor Takahiro Watanabe and Professor Kiyoto Takahata, who spent a lot of time to help me improving this dissertation and preparing the presentation. I also would like to owe my many thanks to Professor Toshinori Sato in Fukuoka University, for his valuable comments and insightful suggestions.

Thirdly, I would like to thank all members in Professor Kimura's Lab. in past and present, who provide a wonderful and positive research environment. They made my life in Japan gorgeous. Especially, I would like to thank Dr. Heming Sun for his comments in my research and Dr. Li Guo for her help in my daily life during my study in Waseda University.

Fourthly, I would like to thank the China Scholarship Council, and the Young Researcher Program by Waseda University and KIOXIA Corporation (former Toshiba Memory Corporation), for their support.

Finally, I conserve my deepest thanks to my family, especially my grandmother in the heaven. It was their love and encouragement that supported me to pursue dreams and never give up.

Contents

Abstract ..... **III**

Acknowledgement ..... **VII**

Contents ..... **IX**

Index of Figures ..... **XI**

Index of Tables ..... **XIII**

List of Abbreviations ..... **XV**

1. Introduction **1**

1.1 Background 1
1.1.1 LSI Implementation of Error-Tolerant Application 1
1.1.2 Approximate Computing 3
1.1.3 Necessity of 8-Bit Approximate Multipliers 6
1.2 Research Status on 8×8 Approximate Multiplier 7
1.2.1 Preliminaries of 8×8 Conventional Multiplier 7
1.2.2 Research Challenge of 8×8 Approximate Multiplier 8
1.2.3 Research Status and Problems 10
1.3 Proposed Concept 12
1.4 Organization of Dissertation 14

2. ASIC-Based Approximate Multiplier using Probability-Driven Inexact Compressors **16**

2.1 Introduction 17
2.1.1 Background 17
2.1.2 Previous Works 18
2.1.3 Research Motivations and Contributions 19
2.2 Proposed Probability-Driven Compressors for Approximate Multiplier 21
2.2.1 Overview of the Proposed Multiplier Design 21
2.2.2 Definition of the Compressor 22
2.2.3 Probability Distribution Analysis 23
2.2.4 Design of Probability-Driven Inexact m:2 Compressor 25
2.3 Approximate Multiplier Design using Proposed Inexact Compressor 31
2.3.1 An 8×8 Multiplier with Different Approximation Levels on 4×4 Multipliers 31
2.3.2 Accumulating Results of Three Blocks in Parallel 32
2.3.3 A Grouped Error Recovery Scheme 33

2.4 Performance Evaluation 34
2.4.1 Evaluation for the Impact of Inexact Compressors 35
2.4.2 Accuracy Evaluation 39
2.4.3 Hardware Analysis 41
2.4.4 Application of Approximate Multipliers to Image Processing 47
2.5 Discussion on Extension to Signed Approximate Multiplier 50
2.5.1 Optimizing the Proposed Inexact Compressors for Signed Multiplier 50
2.5.2 Extension to Signed Approximate Multiplier 55
2.5.3 Experiment for Signed Approximate Multipliers 57
2.6 Summary 61

3. FPGA-Based Approximate Multiplier using Carry-Inexact Elementary Modules 62

3.1 Introduction 63
3.1.1 Background 63
3.1.2 Necessity of FPGA-Based Approximate Multipliers 64
3.1.3 Research Motivations and Contributions 66
3.2 Preliminaries of FPGA-fabric 68
3.3 Proposed Approximate 4×4 Multipliers 70
3.3.1 Occurrence Probability of Carry 71
3.3.2 Approximate 4×4 Multiplier 1 (AFM1) 71
3.3.3 Approximate 4×4 Multiplier 2 (AFM2) 74
3.3.4 Approximate 4×4 Multiplier 3 (AFM3) 76
3.4 Approximate Large Multipliers using Proposed Approximate 4×4 Multipliers as Elementary Modules 79
3.5 Experiment Results and Discussion 81
3.5.1 Experiment Setup 81
3.5.2 Evaluation of 4×4 Multipliers 82
3.5.3 Evaluation of 8×8 Multipliers 83
3.5.4 Image Processing Application 90
3.6 Summary 92

4. Conclusion and Future Work 93

4.1 Conclusion 93
4.2 Future Work 94

Reference 96

Publications 106

Index of Figures

Figure 1-1 Various factors cause inherently error-tolerant feature. 2
Figure 1-2 LSI implementation of applications. 3
Figure 1-3 Comparison of traditional computing and approximate computing. 4
Figure 1-4 Three basic steps in 8×8 conventional multiplier. 8
Figure 1-5 Research challenge of approximate multiplier: trade-off between accuracy loss and hardware saving. 9
Figure 1-6 Research status on ASIC-based approximate multipliers and existing problems. . 11
Figure 1-7 Research status on FPGA-based approximate multiplier and existing problems. . 12
Figure 1-8 Proposed concept: probability-driven carry-restricted compressors. 13
Figure 2-1 Example of approximate approaches in ASIC-based approximate multiplier. (a) Basic strcuture of the 4:2 compressor. (b) Approximate 4:2 compressor by Karnaugh map approximation [43]. (c) Approximate 4:2 compressor by function simplification [50]. . 17
Figure 2-2 Overview of the proposed ASIC-based approximate multiplier. 21
Figure 2-3 Exact 4:2 compressor. (a) Basic architecture. (b) Implementaion. 22
Figure 2-4 Example of partial product matrix of an unsigned 8×8 multiplier. 23
Figure 2-5 Example of partial products in the column of bit 5. 24
Figure 2-6 Probability distribution of the arithmetic sum result of m partial products in an 8×8 multiplier. 25
Figure 2-7 Probability distribution of the arithmetic sum result of m partial products in an image processing. 25
Figure 2-8 Architecture of the m:2 Com. 27
Figure 2-9 Dot notation for Mid_Block, where the proposed inexact compressors are used in partial product accumulation. 32
Figure 2-10 Overall structure of an 8×8 multiplier. 33
Figure 2-11 A grouped error recovery scheme. (a) The function of each group. (b) Overall structure. 34
Figure 2-12 Hardware saving versus error rate for the approximate multipliers with/without proposed probability-driven inexact m:2 Coms. (a) Power saving vs. ER. (b) Area saving vs. ER. (c) Delay saving vs. ER. 36
Figure 2-13 4:2 Compressor (4:2 Com). (a) Logical AND-OR structure. (b) Cell structure after synthesis. 38
Figure 2-14 MRED and PDP for exact multiplier and approximate multipliers. 46
Figure 2-15 Images processd by (a) Exact multiplier. (b) MGER-0g. (c) MGER-1g. (d) MGER-2g. (e) MGER-3g. (f) MGER-4g. 48
Figure 2-16 The trade-off between PDP saving and SSIM degradation for image sharpening application. 49
Figure 2-17 Partial product matrix of an 8-bit signed multiplier. 51
Figure 2-18 The structure of optimized 4:3 Com. 54
Figure 2-19 The structure of 8-bit approximate signed multipliers. Stage 1 is partial product accumulation step, where ● and ○ indicates the partial products generated by AND and NAND gates, respectively. Exact HA and FA are used as CSA step in Stage 2, where ▲

and ■ means the exact and inexact elements, respectively. A CLA is employed in Stage 3. (a) AMSC1: The functionality of m:3 Com is fully used in Stage 1. (b) AMSC2: Only the first and second output bits of m:3 Com are used on low part in Stage 1. (c) AMSC3: Only the first output bit is used on low part in Stage 1. 56
Figure 2-20 Schematic for 4:3 Com. The dotted block with gray background indicates the compound gate cell. 60
Figure 2-21 PDP versus NED for approximate signed multipliers. 60
Figure 3-1 A comparison of ASIC-based implementation and FPGA-based implementation for five state-of-the-art ASIC-based approximate multipliers [48]. 65
Figure 3-2 Overview of the FPGA-based approximate multiplier. 68
Figure 3-3 The structure of 6-input LUT [71]. 69
Figure 3-4 Example of INIT value for LUT6. 69
Figure 3-5 The structure of carry chain [71]. 70
Figure 3-6 Occurrence probability of carry in 4×4 multiplier. 71
Figure 3-7 The structure of AFM1. Layer 1 computes the carry result from the preceding column while Layer 2 generates the sum result for the current column. Layer 3 produces the carry-propagate and carry-generate signals for the carry chain. 72
Figure 3-8 The structure of AFM2. 75
Figure 3-9 The structure of AFM3. Eight LUTs are used to produce the results in parallel. ..77
Figure 3-10 The dot diagrams of three types of proposed 4×4 multipliers. 78
Figure 3-11 Two proposed inexact adders. The dots of ■, ●, ▲ and ⬣ indicates the products from $AL \times BL$, $AL \times BH$, $AH \times BL$, and $AH \times BH$, respectively. (a) IA1: inexact operation is used on columns 4~7, while exact operation is used on columns 8~15. (b) IA2: eight LUTs are used to produce the results in parallel. 80
Figure 3-12 The delay and area of all configurations for the proposed 8×8 multiplier. (a) Delay vs. MRED. (b) Area vs. MRED. The dots on the green line have the best area-MRED tradeoff, and the dots with red circles are selected. 84
Figure 3-13 Accuracy comparison for approximate multipliers. 85
Figure 3-14 Hardware performance of the exact 8×8 multipliers and approximate 8×8 multipliers. (a) Power (b) Latency (c) Area. 86
Figure 3-15 MRED and PDP for 8×8 multipliers. 87
Figure 3-16 Pareto optimal analysis for the 8×8 multipliers. (a) MRED vs. delay. (b) MRED vs. area. 89
Figure 3-17 Processed images by exact multiplier and proposed multipliers. 90
Figure 3-18 PSNR and SSIM values of processed images by approximate multipliers. 91
Figure 3-19 SSIM degradation and PDP saving of all multipliers. 91

Index of Tables

Table 2-1 The behavior of ineact half-adder (inHA).....28
Table 2-2 The behavior of ineact 4:2 compressor (4:2 Com).....29
Table 2-3 The behavior of ineact 6:2 compressor (6:2 Com).....30
Table 2-4 The behavior of ineact 8:2 compressor (8:2 Com).....30
Table 2-5 Five variants of error recovery.....34
Table 2-6 Comparison of synthesized results for inexact compressors.....37
Table 2-7 Accuracy comparisons for approximate multipliers.....40
Table 2-8 Area and delay esitimation.....43
Table 2-9 Synthesized results comparison.....43
Table 2-10 Number of XOR and XNOR gates.....45
Table 2-11 PSNR and SSIM values for the image sharpening application.....49
Table 2-12 Occurrence probability of arithmetic sum results in the signed multiplier 52
Table 2-13 The behavior of 4:3 Com 54
Table 2-14 Accuracy comparisons for signed approximate multipliers 58
Table 2-15 Hardware performance of signed multipliers.....59
Table 3-1 Comparison of DSP blocks and LUTs based implemenations [47].....66
Table 3-2 Input and output configurations for each LUT in AFM1.....73
Table 3-3 The expression of LUTs in AFM1.....73
Table 3-4 Error occurrences of AFM1 and Ca [47].....74
Table 3-5 Input and output configurations for each LUT in AFM2.....75
Table 3-6 The expression of LUTs in AFM2.....76
Table 3-7 Input and output configurations for each LUT in AFM3.....77
Table 3-8 The expression of LUTs in AFM3.....78
Table 3-9 Accuracy comparison of 4×4 multipliers.....82
Table 3-10 Area, latency, power and PDP of 4×4 multipliers.....83
Table 3-11 Configurations for the proposed 8×8 multipliers.....85

List of Abbreviations

ADP	Area-Delay Product
AFM1	Approximate 4×4 Multiplier 1
AFM2	Approximate 4×4 Multiplier 2
AFM3	Approximate 4×4 Multiplier 3
AM1	Approximate Multiplier 1
AMLC	Approximate Multiplier with significance-driven Logic Compression
AMSC	Approximate Multiplier with Sign-focused Compressors
ATC	Approximate Tree Compressor
ATCM	Multiplier with Approximate Tree Compressor
ASIC	Application Specific Integrated Circuit
CLA	Carry Look-ahead Adder
CLB	Configurable Logic Block
CNN	Convolutional Neural Network
CPA	Carry Propagate Adder
CSA	Carry Save Adder
ED	Error Distance
ER	Error Rate
FA	Full-Adder
FPGA	Field Programmable Gate Array
HA	Half-Adder
ISA	Instruction Set Architecture
inHA	inexact Half-Adder
IA1	Inexact Adder 1
IA2	Inexact Adder 2
LUT	Look-Up-Table
LSI	Large Scale Integration
MAC	Multiply-And-Accumulate

MED	Mean Error Distance
MGER	Multiplier with Grouped Error Recovery
MRED	Mean Relative Error Distance
MSE	Mean Squared Error
MUL1	Multiplier 1
MUL2	Multiplier 2
MSB	Most Significant Bit
m:3 Com	m to 3 Compressor
NED	Normalized Error Distance
NMED	Normalized Mean Error distance
NWCE	Normalized Worst Case Error distance
PDP	Power-Delay Product
PSNR	Peak Signal-to-Noise Ratio
RED	Relative Error Distance
R4ABM	Radix-4 Approximate Booth Multiplier
SH	Sum result of High block
SL	Sum result of Low block
SM1	Sum result of Middle block-1
SM2	Sum result of Middle block-2
SSIM	Structural Similarity
TPU	Tensor Processing Unit
UDM	Underdesigned Multiplier
VOS	Voltage Over Scaling
WCRE	Worst Case Relative Error
4:2 Com	inexact 4:2 Compressor
6:2 Com	inexact 6:2 Compressor
8:2 Com	inexact 8:2 Compressor

1. Introduction

1.1 Background

1.1.1 LSI Implementation of Error-Tolerant Application

Despite the advancement in semiconductor technologies and development of computational system techniques, the overall energy consumption of large-scale applications is still rapidly growing due to an ever-increasing amount of information. Today, lots of computation-massive applications such as image processing, machine learning, data mining/analytics, web search and wireless communication are used more and more. Their computational and storage demands on modern systems have far exceeded the available resources. In addition, the electricity consumption is increasing of the data center which store and run these applications. As reported in [6], the electricity consumption of just US data centers is estimated to increase from 61 billion kWh in 2006 to 140 billion kWh in 2020. This not only brings the fiscal expenditure but also the environment issues. It trends to pose severe technology challenges, such as energy efficiency, circuits reliability, and high performance. Therefore, there is a genuine need to improve the resource efficiency for these emerging workloads in order to reduce the energy consumption and to keep pace with the growth of information that needs to be processed.

Fortunately, such computation-massive applications usually have an inherently error-tolerance property, that is, they don’t require fully exact computation results. In general, these applications are demonstrated and defined as error-tolerant applications [1][2], such as data mining, robotics, search, image processing, pattern recognition and so on. Error-tolerant applications are very common and close to human.

This inherent feature of error-tolerant applications arises due to following factors, as shown in Figure 1-1: (i) For the applications such as search and classification, they

usually return a range of answers, rather than a unique and golden output. This means multiple answers are equally suitable for users. Therefore, it is acceptable that very few inexact outputs and most exact outputs are in a wide range of outputs. For other applications, even a completely prefect result exists, the algorithms in applications may not be able to find it, because it might cause a huge hardware effort. (ii) Besides, the perceptual limitation of humans determines that the inexact yet good enough result is acceptable. The common examples are most machine learning, image processing and recognition applications. The tiny change in pixel value or similar effects may not be perceived by humans due to their psycho-visual limits. (iii) In addition, noise usually exists in real input data, which is naturally propagated to the intermediate and final results. This is the robustness of the application. In another word, the robustness to noise in traditional paradigm provides the robustness to approximation. (iv) In computation-massive applications, most algorithms used have redundant set and self-healing property, such as aggregation and iterative-refinement. This feature could recover the accuracy loss by approximations.

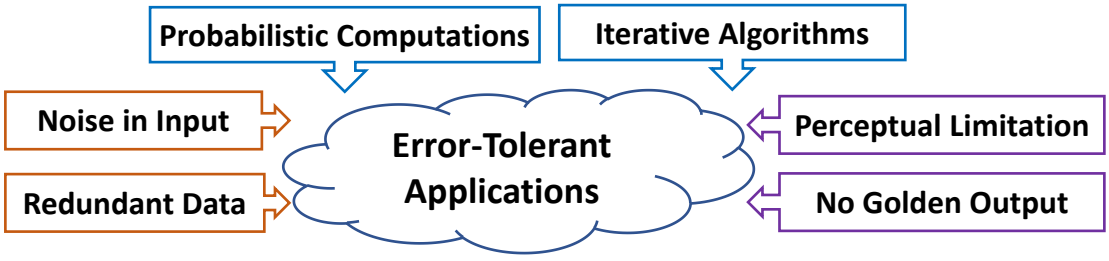


Figure 1-1 Various factors cause inherently error-tolerant feature.

Take image processing application as an example, occasional errors such as dropping a particular pixel or a small image quality loss rarely affect user's satisfaction. Human eyes cannot recognize the missing information and still correctly perceive an approximately processed image. This is caused by the perceptual limitation of humans and noise or redundant data in image.

For good product of applications, large scale integration (LSI) implementation is important because of huge computation complexity in applications and limited

resources in hardware. However, most original approaches designing applications are software oriented, which didn't consider the LSI implementation. Generally, the performance of circuits has been improved as the guidance of Moore's law [3] and Dennard's scaling [4]. As Dennard's scaling tends to an end, the performance improvement is slow and may stop in near future. Therefore, LSI implementation is very important. This dissertation focuses on optimized LSI implementation or hardware approaches.

To implement applications into VLSI, there are several steps as shown in Figure 1-2 including VLSI oriented algorithm, designing hardware architecture and LSI implementation. Among hardware architecture, multiplier is very important because basic units of lots of applications are implemented by multiplier and adder. Multiplier usually costs more hardware than adder. Therefore, optimized multiplier is the focus of this dissertation. By optimizing the basic unit of lots of applications, it is promising to reduce the energy consumption of the application and further reduce the resource cost of the whole systems or data centers.

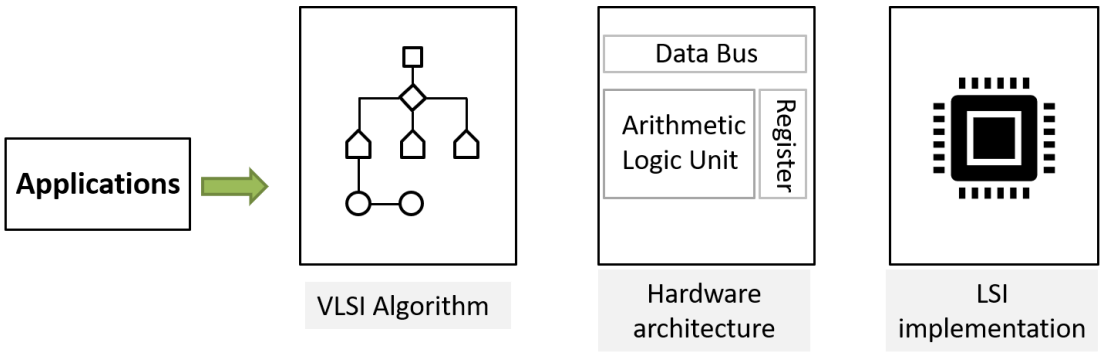


Figure 1-2 LSI implementation of applications.

1.1.2 Approximate Computing

During the whole design process of applications, there are many factors need to be considered, such as accuracy or quality and hardware performance like power, delay and circuit size. The balance of these factors is necessary. Approximate computing is one approach to trade accuracy to hardware saving, which is motivated by efficient

hardware implementation and exactness relaxation in error-tolerant applications. This computing technique returns an inexact result by skipping or simplifying some operations. It can reduce hardware consumption.

Approximate computing can be applied inside digital computers, where the inexact computations have been done with respect to inputs with noise and for users who have the perception limitation. Figure 1-3 illustrates the comparison of traditional computing and approximate computing. The accuracy is decreased with approximate computing, but the power efficiency and performance can be improved. This methodology has attracted a surge of interest from both academic and industry [5]-[7], as a promising technique to achieve diverse optimizations, such as energy saving, smaller design area and high speed.

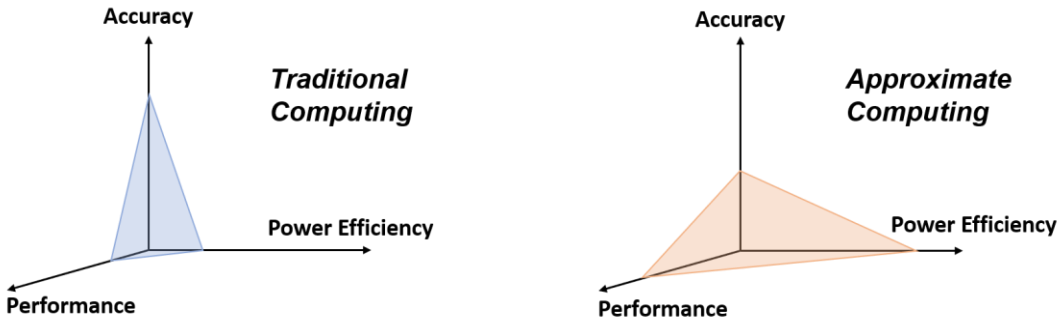


Figure 1-3 Comparison of traditional computing and approximate computing.

Approximate computing can be applied at software, architecture and circuit levels:

At software or algorithm level, one of the most efficient approximation approaches is precision scaling [8]-[10]. This technique alters the precision (bit-width) of the input to shorten the width of an operand utilized in the applications. Skipping computation is another popular technique which works by skipping some iterations of a loop to reduce the execution time and save the resources [11][12].

At architecture level, there are many approximate computing techniques are explored, such as approximate memories [13][14], approximate computing in programmable processors [15], approximate accelerators [16], and approximate instruction set architecture (ISA) [17].

At circuit level, approximate arithmetic units [18]-[21], approximate circuit synthesis [22][23] and voltage over scaling (VOS) [24][25] are the commonly studied techniques. For a given circuit, energy consumption could be saved by reducing the logic gates, approximating the functions and lowering its supply voltage. VOS is a technique reducing the supply voltage, it is one of the most direct techniques to save power consumptions, especially dynamical power consumptions. However, this technique has a very high implementation cost to control voltage, because it requires carefully allocate higher supply voltage to critical part to ensure the accuracy, along with lower the supply voltage for the less significant part. Consequently, the majority of approximate circuits are designed on approximate arithmetic units.

Many researches from academic area and industries have been show the feasibility of approximate computing [26]-[30]. Convolution neural network (CNN) targets brain-like functionality and is based on a simple artificial neuron. Its basic operation is convolution where the input is multiplied and accumulated with the weight. Recently, CNNs could achieve high accuracy for many tasks. However, CNNs usually require significant computational resources, along with the huge hardware consumption. To effectively train and use CNNs, IBM and Google have exploited approximate computing for CNNs [26]-[28]. Especially, the Tensor Processing Unit (TPU) [28] which changes the floating-point number to 8-bit fixed-point number, has been successfully used for Google photo, Google translation and AlphaGo. In addition, approximate CNNs with optimized computation units (e.g. approximate multiplier) have been tested the effectiveness of approximate computing. For example, for image classification, in [30], the CNNs with approximate computing have been introduced by using approximate multiplier. The conventional multiplier in quantized CNN is changed to two short-size multipliers with one inexact adder. The quantized CNN on ImageNet dataset was evaluated. The accuracy loss is 3%, but the latency saving is 17% and power saving is 15%. For the applications like handwritten digit recognition, [27] has proposed approximation methods for CNNs to do this task. The floating-point number in CNN is changed to 16-bit fixed-point number. For the approximate LeNet-5 on MNIST dataset, the accuracy loss is only 0.07%. For the application like web search,

PageRank is a typical method, [29] introduced approximation for this application by altering the mantissa width in double floating-point number (i.e. 52-bit mantissa to 16-bit mantissa). Such researches on CNNs and applications show the feasibility of approximate computing.

1.1.3 Necessity of 8-Bit Approximate Multipliers

Among three commonly studied levels (software/algorithm level, architecture level and circuit level) of approximate computing, approximation at circuit level has recently attracted more and more attentions. This is because in error-tolerant applications, multiply-and-accumulate (MAC) operations $\sum_i X_i * Y_i$ is widely used where X_i is weight and Y_i is input. For example, 90% of the computation in convolution operation are MAC operations; filtering an image of 512×512 pixels costs 262144 MAC operations [31]. The basic computations of MAC operations are multiplication and addition. Therefore, approximation applied at addition and multiplication plays a pivotal role in determining the performance of many computation-massive applications. The energy efficiency of this level could contribute to the overall system. More importantly, the quality losses arose by the approximation at circuit level is more controllable than that at other levels.

As the key and basic circuit, approximate adders both at gate level and transistor level have been extensively studied [32]-[35]. In general, approximate adders can be classified into two types: segmented adder and non-segmented adder. They both mainly reduce the carry propagation delay, while the power and area reductions are slight. In addition, approximate adders have relatively high accuracy loss, because the error in their simple structures would have a significant impact to overall result.

Another important arithmetic unit, multiplier, has become hot and principal research domain in approximate computing [36]-[55]. As the basic arithmetic unit, a multiplier is more energy-hungry than an adder, hence multipliers are characterized by high-complexity logic design and have high energy consumption. In addition, a lot of 8-bit multiplications are utilized in applications for manipulating images such as

filtering operations and classification using convolutional neural networks. Moreover, 8-bit multipliers are also used to construct larger multiplier as basic modules. Therefore, 8-bit approximate multiplier design has been a focus of approximate circuit, and it is vital to propose low-cost approximate multiplier under insignificant accuracy loss.

1.2 Research Status on 8×8 Approximate Multiplier

1.2.1 Preliminaries of 8×8 Conventional Multiplier

Generally, a multiplier consists of three steps, as shown in Figure 1-4: partial product generation, partial product accumulation and final carry propagate adder. Partial product accumulation step occupies the dominated energy consumption and circuit complexity in a multiplier.

In partial product generation, partial product matrix including 64 partial products is generated by AND operation. Then, 8 rows of partial product matrix are accumulated and reduced into 2 rows by adders or compressors. In this step, compressors are key components to accumulate partial products, which are defined as the circuits to compressor/reduce several elements to less elements. Half adder and full adder are usual compressors in exact multipliers, which is regarded as 2 to 2 compressor and 3 to 2 compressor, respectively. For example, a full adder can add (compress) three partial products and generate 1-bit result (i.e. sum) at the same position with input, along with another bit result (i.e. carry) at 1-bit higher position. By repeatedly applying such compressors, 2-bit results at each column are computed. Finally, carry propagate adder processes 2 rows and produces 16-bit product.

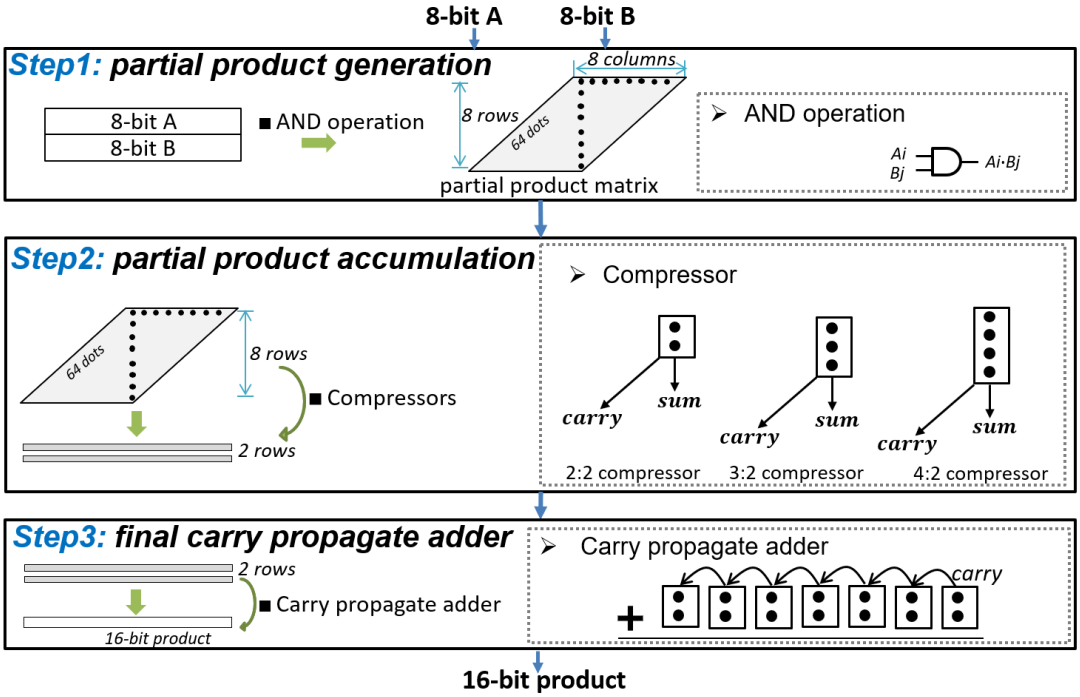


Figure 1-4 Three basic steps in 8×8 conventional multiplier.

1.2.2 Research Challenge of 8×8 Approximate Multiplier

Approximate multiplier is designed based on the three conventional steps as introduced in the last section. Recently, there are two major platforms to run designed multiplier: one is ASIC (Application Specific Integrated Circuit) and the other is FPGA (Field Programmable Gate Array). Both platforms have special features and multiplier circuits need to be optimized for each platform. Depending on platform, approximate multipliers are called as ASIC-based approximate multiplier and FPGA-based approximate multiplier. They both are of great importance with worth deep investigation. In ASIC-based designs, logic gates are deployed for the implementation of different logic circuits. For approximation, circuits could be simplified by using simpler logic gates. This is the most intuitive methodology to approximate the function of one multiplier, hence ASIC-based approximate multiplier is an attractive topic both now and in the future. On the other hand, FPGA has emerged as a potential platform to accelerate amount of computations, because of its short turnaround time. The

architectural difference between ASIC and FPGA determines that the ASIC-based multiplier design is not effective to FPGA. FPGA-specific approximate multipliers have been a hottest research domain for recent years. Thus, both of ASIC-based approximate multiplier and FPGA-based approximate multiplier are extremely necessary.

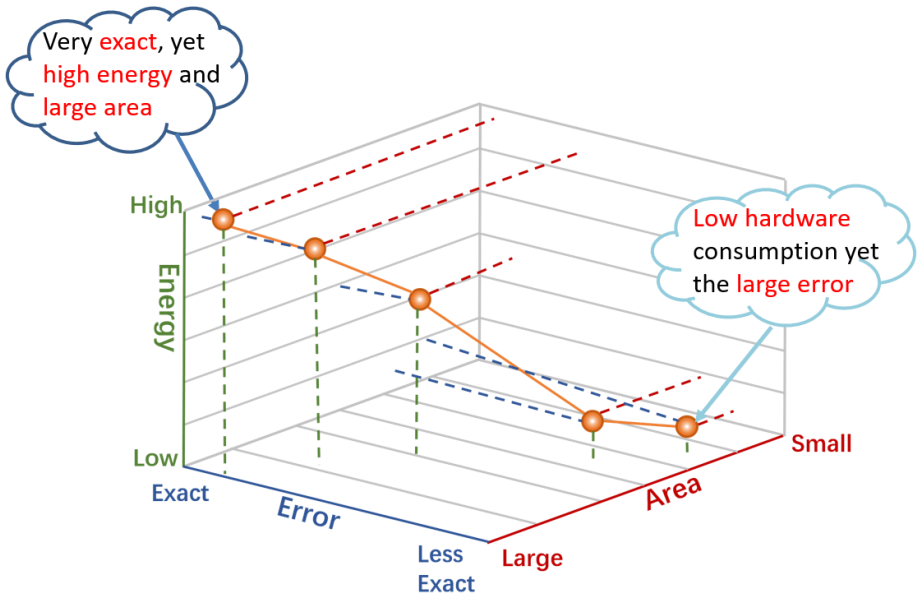


Figure 1-5 Research challenge of approximate multiplier: trade-off between accuracy loss and hardware saving.

ASIC-based approximate multiplier and FPGA-based approximate multiplier both face a common challenge, how to trade off the accuracy loss with hardware saving. This is determined by the nature of approximate computing. Figure 1-5 shows the rough trade-off relation among energy, error and area. An exact multiplier has error of 0 but its energy is high and area is large. Depending on the acceptable error, approximate multipliers can reduce energy and area as shown in the figure. For example, a design may be very exact yet with a high energy consumption or large design area; or a very energy-efficient design may have an extremely low accuracy. It is difficult to achieve the best result of the trade-off, but it is still an urgent requirement to propose an approximate design, which achieves power, area and delay savings under insignificant accuracy loss.

1.2.3 Research Status and Problems

The investigations of approximate multipliers have been carried on for recent years, and many significant researches have been made for approximate multipliers. The majority of researches focused on the second step (i.e. partial product accumulation).

The following shows a brief review of previously proposed approximate multipliers for ASIC platform and FPGA platform and their remaining problems.

ASIC-based approximate multipliers: Circuit functions can be implemented and mapped by logic gates on ASIC platform. By simplifying the logic function, the complexity of circuit can be reduced. This is the most intuitive methodology, because it can foresee the low complexity of circuit from the simplification/approximation of the logic function. As pointed out in Section 1.2.1, the partial product accumulation step has the primary resource consumption in multiplier. This is mainly caused by the compressors, which are characterized as XOR-rich circuits. Thus, there are many researches on ASIC-based multiplier with approximate compressors [39][45-47][50][52-54]. As illustrated in Figure 1-6, those works can be concluded into two categories: propagation restriction, and less-XOR computation. As for propagation restriction, there are two methods. The first method is to use OR gate to extremely approximate the compressor. The 2-input operation is focused by [39] and 3-input operation is focused by [53]. The second method is to divide the carry to two inexact parts [54]. The operations of two parts are in parallel. As for the method category of computation with less XOR gate, the operations of exact 2-input, 3-input and 4-input compressors are simplified by reducing XOR gates. For example, some terms including XOR operation in the function are deleted.

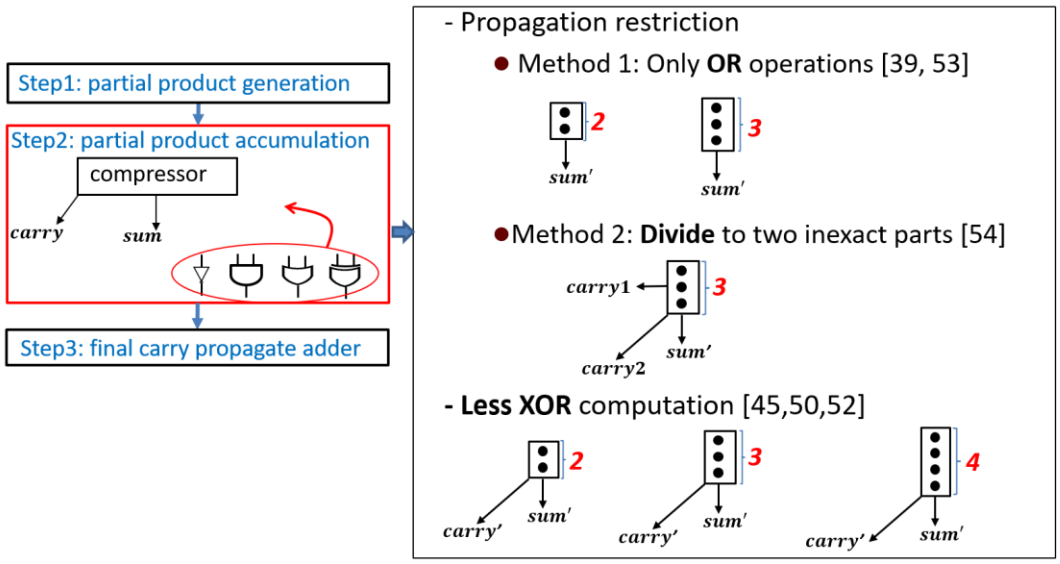


Figure 1-6 Research status on ASIC-based approximate multipliers and existing problems.

Existing issue: Approximate compressors have been proposed in most approximate multipliers to accumulate partial products. They all can achieve hardware reduction, while they still have some issues. The input size of previous approximate compressors is limited to 4-bit. To accumulate the fixed elements, the smaller compressors lead to more accumulation stages, which further limit the potential of hardware reduction. In addition, most of previous approximate compressors still include XOR gates, which is difficult to be synthesized.

FPGA-based approximate multipliers: Recently, FPGA has become a promising choice for computing systems because of its reconfigurability and fast development time. For computation-massive applications on FPGA, it is vital to explore the FPGA-based approximate multipliers by considering look-up-tables (LUTs) and special carry chain. For recent years, it has been the newest and hottest topic for many applications and attracted more and more attentions [46]-[48]. The focus of FPGA-based approximate multiplier is also in Step 2 (i.e. partial product accumulation), while the compressors in this step are implemented by LUTs and carry chain. Figure 1-7 views the recent state-of-the-art FPGA-based approximate multipliers. One method is introduced in [46], where the carry-in result from preceding bit is inexactly predicted

and used as one input for compressor. Thus, the computation of each position is independent with each other. The same author discussed another approximate multiplier in [47] by omitting one input in the compressor to save LUTs.

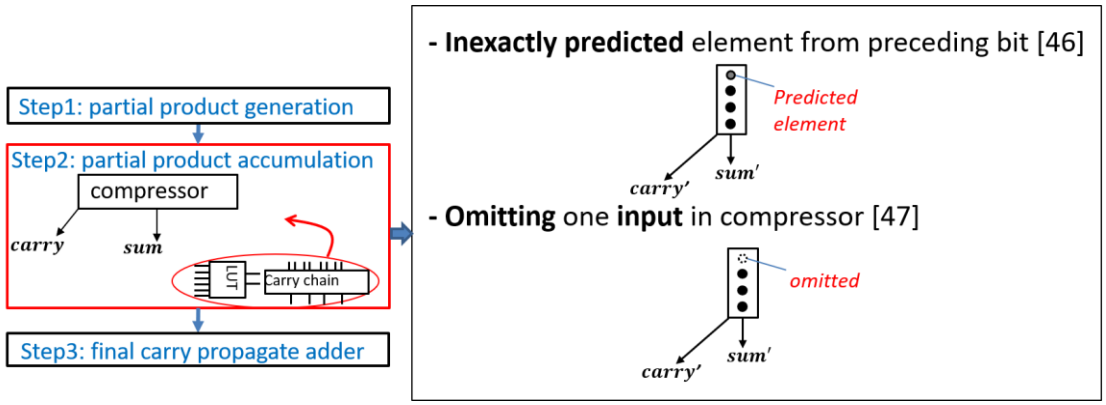


Figure 1-7 Research status on FPGA-based approximate multiplier and existing problems.

Existing issue: The existing methods could save the FPGA resource by approximation, but there is one common issue for the state-of-the-art FPGA-based approximate multipliers. The approximation (i.e. inexactly predicted carry-in and omitting one input) on the input of compressor affects both of two outputs of the compressor. It incurs the high accuracy loss.

1.3 Proposed Concept

To achieve the target of approximate multiplier (i.e. trade-off between accuracy and hardware), and to solve the existing limitations in approximate multipliers, this dissertation proposes probability-driven carry-restricted compressors for two important types of approximate multipliers, ASIC-based approximate multiplier and FPGA-based approximate multiplier. The compressors are the main units of multiplier, and they cost the primary hardware in the multiplier. Therefore, new compressor design is proposed in this dissertation. By considering the low occurrence probability of carry in the compressor, the approximation is applied on the carry. The proposed method could save

the hardware of the compressor and ensure that the accuracy loss is small as well (due to low occurrence probability of carry).

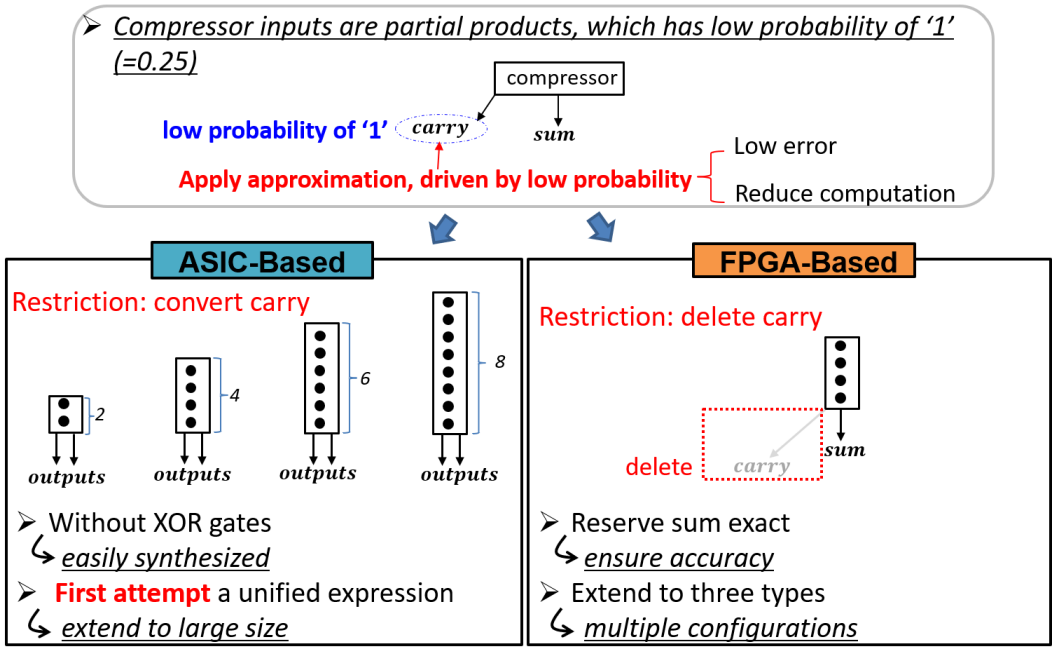


Figure 1-8 Proposed concept: probability-driven carry-restricted compressors.

As for ASIC-based multiplier, in the accumulation/compression step, compressor is implemented by logic gates. It usually generates the carry to higher position. The carry costs the hardware yet occurs in rare cases. By caring about the low 1-probability of one partial product and further the low 1-probability of carry, new compressors have been developed to generate 2-bit outputs at the same position by restricting (i.e. converting) the carry to the position as the sum. Different from small compressors (i.e. the largest size is 4-bit) in previous approximate multiplier, this proposal is the first attempt to introduce a unified expression to extend to large compressor (i.e. the largest size is 8-bit). It solves the problem of slight hardware saving by more compressor stages caused by small compressors. Moreover, the proposed compressor design does not include the XOR gates while most of previous works still include XOR gates. XOR gates are most energy-consuming gates among logic gates in ASIC circuit. Therefore, by avoiding XOR gates, the proposed compressor is easily and effectively to be synthesized and ASIC circuit can be simplified.

As for FPGA-based approximate multiplier, the compressors in accumulation step are implemented by LUTs. One carry computation with FPGA-fabric usually costs one corresponding LUT. By considering that the occurrence probability of carry from one bit to higher one bit is low, the computation of carry result is omitted, along with the sum result is computed in the exact manner. Such compressor is without carry. By doing so, the LUT resource for carry can be saved and the low error can be guaranteed at the same time. To provide multiple choices for FPGA's reconfiguration feature, difference approximation degrees are applied where three proposed multipliers are implemented with different number of no-carry compressors.

1.4 Organization of Dissertation

The contents of this dissertation are organized as follows.

In Chapter 1, the background of this dissertation is introduced, including error-tolerant applications, approximate computing and the necessity of 8×8 approximate multipliers. The research states and existing problems on approximate multipliers are stated. To achieve the trade-off between accuracy loss and hardware saving, approximate multipliers with probability-driven carry-restricted compressors are proposed.

In Chapter 2, an ASIC-based approximate multiplier design is proposed, which is based on a novel probability-driven inexact compressor design by focusing restriction on the carry. The proposed compressor methodology reduces the height of partial product matrix to two rows with one stage. To compensate accuracy loss, a grouped error-recovery scheme is proposed to produce the final product. Such grouped error recovery is the derivate of probability-driven compressor. To demonstrate the feasibility that the proposed unsigned approximation technique can be extended to signed integer operation, signed approximate multiplier with optimized probability-driven inexact compressor is also discussed.

In Chapter 3, an FPGA-based approximate multiplier design is introduced. This FPGA-based approximate multiplier is proposed by carefully considering the structure

of FPGA (i.e. LUT and special carry chain). The proposed approximate multiplier is constructed from four optimized 4×4 multiplier. In each 4×4 multiplier, the compressor is implemented by LUTs. By considering the low occurrence probability of carry, the no-carry compressor is proposed to accumulate partial products. Three types of 4×4 multipliers are introduced. To sum four 4×4 approximate multipliers, two types of inexact adders are proposed. To provide multiple configurations of FPGA-based approximate multipliers, all possible combinations from elementary modules are discussed in this chapter.

In Chapter 4, this dissertation is concluded and some future works are given.

2. ASIC-Based Approximate Multiplier using Probability-Driven Inexact Compressors

In this chapter, ASIC-based approximate multiplier design is to be discussed.

Section 2.1 detailly states the research status and problems on ASIC-based approximate multipliers, and the motivations and contributions of this work also are introduced in this section.

A novel probability-driven inexact compressor design is proposed in Section 2.2, which is based on the probability distribution and restricts the carry. This compressor design is utilized in the multiplier to accumulate partial products and the height of partial product matrix is reduced to two rows.

After the design of inexact compressors, approximate multiplier with the inexact compressors is proposed in Section 2.3. To compensate the accuracy loss of the multiplier, a grouped error-recovery scheme is exploited to achieve different levels of accuracy.

Section 2.4 presents the experiment results in terms of accuracy evaluation and hardware performance for the exact multiplier, existing state-of-the-art approximate multipliers and proposed multiplier. In terms of mean relative error distance (MRED), the accuracy losses of the proposed multipliers are from 1.07% to 7.86%. Compared with the exact multiplier using 40nm process, the most accurate variant of the proposed multipliers can reduce power by 59.75% and area by 42.47%. The critical path delay reduction is larger than 12.78%. The proposed multiplier design has a better accuracy-hardware result than other designs with a comparable accuracy, which achieves the target of this research.

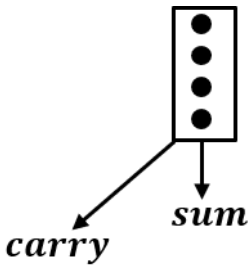
In Section 2.5, the proposed restricted carry generation and propagation approach is extended to signed approximate multiplier, which achieves low energy consumption under the similar accuracy loss with other existing signed approximate multipliers.

Section 2.6 concludes the ASIC-based approximate multiplier.

2.1 Introduction

2.1.1 Background

As introduced before, approximate computing has been considered as a potential approach to achieve significant reduction in energy cost by exploiting the exactness relaxation in error-tolerant applications, while still produces sufficiently exact results. ASIC-based approximate multiplier design is one of the most attractive topics, because it provides a fine granularity control of the circuits at gate level. There are many commonly exploited techniques in ASIC-based approximate multiplier, such as function simplification, Karnaugh map approximation and truncation.



(a)

x_3x_4 \ x_1x_2	00	01	11	10
00	00	01	10	01
01	01	10	11	10
11	10	11	100 → 11	11
10	01	10	11	10

(b)

$$sum = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus C_{in}$$
$$carry = (x_1 \oplus x_2 \oplus x_3 \oplus x_4) \cdot C_{in} + \overline{(x_1 \oplus x_2 \oplus x_3 \oplus x_4)} \cdot x_4$$
$$C_{out} = (x_1 \oplus x_2) \cdot x_3 + \overline{(x_1 \oplus x_2)} \cdot x_1$$

$$sum' = (x_1 \oplus x_2) + (x_3 \oplus x_4) + x_1x_2x_3x_4$$
$$carry' = x_1x_2 + x_3x_4$$

(c)

Figure 2-1 Example of approximate approaches in ASIC-based approximate multiplier. (a) Basic strcuture of the 4:2 compressor. (b) Approximate 4:2 compressor by Karnaugh map approximation [43]. (c) Approximate 4:2 compressor by function simplification [50].

Take the key component (i.e. 4:2 compressor) in the multiplier as an example to introduce the approximation techniques for ASIC-based multiplier. The structure of accumulation with 4:2 compressor in the multiplier is shown in Figure 2-1 (a).

Karnaugh map approximation considers the output which has the different bit-width, to change the exact output to the smaller output, as shown in Figure 2-1 (b). Figure 2-1 (c) shows function simplification which usually ignores some items in the function to simplify the expression. Another approach is truncation, which usually skips or ignores the least significant part in the multiplier [51].

The function of a multiplier can be directly approximated to map the logic gates and design the optimized circuit. Therefore, most of existing researches focused on the ASIC-based approximate multiplier design.

2.1.2 Previous Works

As introduced in Section 1.2,1, a multiplier usually includes partial product generation, partial product accumulation and final carry propagate adder. The step of partial product accumulation occupies the primary hardware consumption and circuit complexity in a multiplier, hence most of studies focus on the approximation of this step.

Here is detailed review of existing ASIC-based approximate multipliers related to this research. The methods of previous works could be concluded as two categories: propagation restriction [39, 53, 54] and computation with less XOR gates [45, 50, 52]. The propagation restriction with OR operation is discussed in [39] and [53]: In [39], a significance-driven logic compression for approximate multipliers (AMLC) is discussed, where this compression uses 2-input OR operation to replace the exact half adder. 3-input OR gates used in LSB part as propagation restriction is introduced in [53]. Propagation restriction by dividing carry is discussed in [54], where the carry of compressor is divided to two semi-carries for parallel generation. As for the less-XOR computation, inexact 4-input compressor with 2 XOR gates is introduced in [45]. In [50], two approximate multipliers (MUL1 and MUL2) are explored using inexact half-adder, full-adder and 4:2 compressors by deleting the terms including XOR operation in the function. In [52], an approximate tree compressor (ATC) constructed from incomplete adder cells is introduced to reduce the partial product matrix by half,

resulting in two approximate multipliers with ATC, they are named as ATCM1 and ATCM2.

To compensate the accuracy loss of overall multiplier, error recovery strategies have been discussed. In [45], an inexact half-adder is proposed to accumulate partial products in parallel; a conventional adder is used to recovery error of approximate multipliers (AM1 and AM2). The 4:2 compressor with an error recovery modular is proposed and used in multiplier in [40].

A reconfigurable construction is another approach for accumulating partial products. In [55], an inexact 2×2 multiplier is proposed by simplifying its Karnaugh-Map expression and used in larger multipliers. Approximate 4:2 compressors are introduced in [43] by approximating the Karnaugh map, then compressors are utilized in the multiplier to accumulate partial products.

In general, compressors are XOR-rich circuits in the multiplier, and XOR gate costs more power and area than AND and OR gates. High energy consumption in a multiplier is usually caused by compressors with a lot of XOR gates. The inexact compressors have been introduced in previous works. However, they still have high complexity because of the remaining XOR gates. In addition, above previous works just discussed compressors with the input width up to 4-bit, which incurs more compressor stages than large-size compressors.

Error-recovery strategy is important to compensate accuracy loss. However, previous works employed conventional adders to compensate error bit by bit. It usually causes the hardware overhead and extra delay.

2.1.3 Research Motivations and Contributions

Motivated by the demand of considerable trade-off between hardware saving and accuracy loss, this research aims to achieve better energy saving than existing works under the same accuracy loss. To further reduce the hardware consumption in the multiplier, this research explores large-size compressors without XOR gates. To compensate accuracy loss, error recovery is an important approach.

In this research, an approximate multiplier design is proposed by using probability-driven compressors with only AND and OR gates (no XOR gates), which can be synthesized to energy- and area-efficient cells, compound gates. In addition, a unified expression is proposed to extend the input of compressor to larger size (8-input), while the previous approximate multipliers focus on small compressor (i.e. largest size is 4-input). Different from the propagation restriction with only OR gates in [39, 53] and two semi-carries in [54], the proposal considers the low probability of carry and converts it to the position with the sum to restrict the upper carry. By caring the probability of carry, the proposed method could save the hardware and ensure the accuracy. To compensate the accuracy loss of the multiplier, a grouped error recovery scheme is proposed. Different from bit-wise error-recovery approach, this proposed scheme processes the error compensation elements in the form of group for reducing critical path. Area and delay of the proposed multipliers are estimated theoretically. Moreover, accuracy evaluation and circuit simulation are provided. To test the validity of the approximate multipliers, image sharpening is considered. The primary contributions of this research are as follows:

i) A novel **probability-driven inexact m:2 compressor design** is proposed without XOR gates to accumulate m partial products into 2 bits (m = 2, 3, …, 8). There needs only one compressor stage before the final addition. Besides, a general expression for m:2 compressor is conducted to extend the input to larger size.

ii) A new approximate structure for 8×8 multiplier is introduced, which is divided into four 4×4 multipliers with different precision operations. The highest 4×4 multiplier is exact and OR operation is used on the lowest one, while the proposed compressors are used on middle two multipliers.

iii) **A grouped error recovery scheme** is proposed to compensate accuracy loss. The proposed scheme processes error elements in the form of group which can shorten the delay. Five variants of error recovery are presented for different accuracy requirements.

2.2 Proposed Probability-Driven Compressors for Approximate Multiplier

2.2.1 Overview of the Proposed Multiplier Design

A 2n×2n multiplier (denoted as $A \times B$) can be divided into four n×n multipliers as described by

$$A \times B = (A_H \times 2^n + A_L) \times (B_H \times 2^n + B_L)$$
$$= A_H \times B_H \times 2^{2n} + (A_H \times B_L + A_L \times B_H) \times 2^n + A_L \times B_L. \quad (2\text{-}1)$$

where, A_H (B_H) and A_L (B_L) are an upper- and a lower-half of A (B), respectively. By using this structure, an 8×8 multiplier can be built from four 4×4 multipliers.

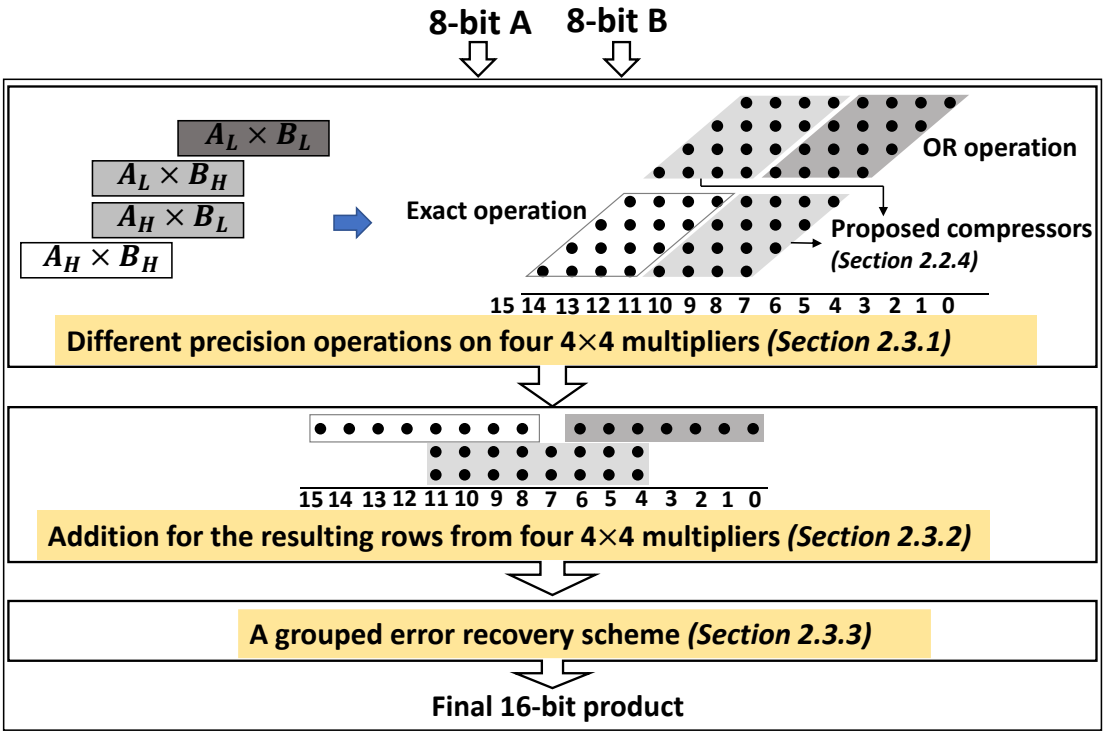


Figure 2-2 Overview of the proposed ASIC-based approximate multiplier.

The overview of the proposed approximate multiplier design is illustrated in Figure 2-2. A multiplier is divided into four small-size multipliers. Approximation are applied

on multipliers of $A_L \times B_H$, $A_H \times B_L$ and $A_L \times B_L$, along with keeping the highest block as exact. This allocation approach trades the accuracy for the hardware reduction. To accumulate the partial products in multipliers of $A_L \times B_H$ and $A_H \times B_L$, a novel inexact compressor design is proposed in Section 2.2.4. This is another and important technique to achieve the good balance among hardware performance and accuracy quality. Section 2.3.1 detailly introduces the different precision operations applied on four multipliers, based on the significances of each multipliers. To improve accuracy, a grouped error-recovery scheme is proposed in Section 2.3.3 and used in the final step of a multiplier.

2.2.2 Definition of the Compressor

Compressor is a circuit that reduces several operands into less operands. In a multiplier, compressors are used to accumulate several rows of partial products into two rows. It is worth mentioning that, a full-adder processes 3 elements and generate 2 outputs, hence it is generally called as a 3:2 compressor.

To lower the latency of partial product accumulation step, large-size compressors are also widely used in the multiplier, such as 4:2 compressor. Figure 2-3 shows the structure of a 4:2 compressor [42].

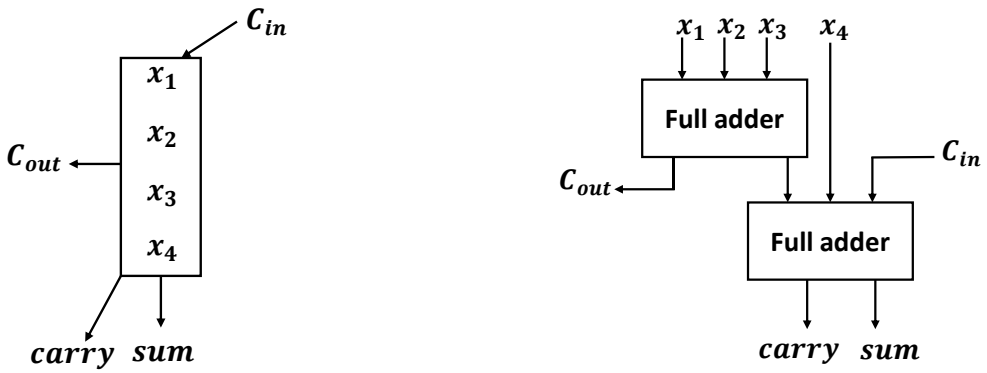


Figure 2-3 Exact 4:2 compressor. (a) Basic architecture. (b) Implementaion.

The function of a 4:2 compressor is given by

$$sum = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus C_{in},$$
$$C_{out} = (x_1 \oplus x_2) \cdot x_3 + \overline{(x_1 \oplus x_2)} \cdot x_1,$$
$$carry = (x_1 \oplus x_2 \oplus x_3 \oplus x_4) \cdot C_{in} + \overline{(x_1 \oplus x_2 \oplus x_3 \oplus x_4)} \cdot x_4 \quad (2\text{-}2)$$

where '+' indicates OR operation and '·' means AND operation.

The output sum has the same weight with four inputs x_1, x_2, x_3, x_4, while the output $carry$ is weighted one-bit-higher position. A 4:2 compressor receives a carry-in signal C_{in} from the preceding module on one-bit-lower position and produces a carry out signal C_{out} to the next one-bit-higher module. A 4:2 compressor is usually implemented with two serially connected full adders, as shown in Figure 2-3 (b).

2.2.3 Probability Distribution Analysis

Consider two unsigned n-bit operands $\alpha = \sum_{i=0}^{n-1} 2^i \times \alpha_i$ and $\beta = \sum_{j=0}^{n-1} 2^j \times \beta_j$. The partial product is the result of AND operation of the bits of α_i and β_j. Figure 2-4 shows the partial product matrix of an unsigned 8×8 multiplier.

								×	α_7	α_6	α_5	α_4	α_3	α_2	α_1	α_0
									β_7	β_6	β_5	β_4	β_3	β_2	β_1	β_0
									$\alpha_7 \cdot \beta_0$	$\alpha_6 \cdot \beta_0$	$\alpha_5 \cdot \beta_0$	$\alpha_4 \cdot \beta_0$	$\alpha_3 \cdot \beta_0$	$\alpha_2 \cdot \beta_0$	$\alpha_1 \cdot \beta_0$	$\alpha_0 \cdot \beta_0$
								$\alpha_7 \cdot \beta_1$	$\alpha_6 \cdot \beta_1$	$\alpha_5 \cdot \beta_1$	$\alpha_4 \cdot \beta_1$	$\alpha_3 \cdot \beta_1$	$\alpha_2 \cdot \beta_1$	$\alpha_1 \cdot \beta_1$	$\alpha_0 \cdot \beta_1$	
							$\alpha_7 \cdot \beta_2$	$\alpha_6 \cdot \beta_2$	$\alpha_5 \cdot \beta_2$	$\alpha_4 \cdot \beta_2$	$\alpha_3 \cdot \beta_2$	$\alpha_2 \cdot \beta_2$	$\alpha_1 \cdot \beta_2$	$\alpha_0 \cdot \beta_2$		
						$\alpha_7 \cdot \beta_3$	$\alpha_6 \cdot \beta_3$	$\alpha_5 \cdot \beta_3$	$\alpha_4 \cdot \beta_3$	$\alpha_3 \cdot \beta_3$	$\alpha_2 \cdot \beta_3$	$\alpha_1 \cdot \beta_3$	$\alpha_0 \cdot \beta_3$			
					$\alpha_7 \cdot \beta_4$	$\alpha_6 \cdot \beta_4$	$\alpha_5 \cdot \beta_4$	$\alpha_4 \cdot \beta_4$	$\alpha_3 \cdot \beta_4$	$\alpha_2 \cdot \beta_4$	$\alpha_1 \cdot \beta_4$	$\alpha_0 \cdot \beta_4$				
				$\alpha_7 \cdot \beta_5$	$\alpha_6 \cdot \beta_5$	$\alpha_5 \cdot \beta_5$	$\alpha_4 \cdot \beta_5$	$\alpha_3 \cdot \beta_5$	$\alpha_2 \cdot \beta_5$	$\alpha_1 \cdot \beta_5$	$\alpha_0 \cdot \beta_5$					
			$\alpha_7 \cdot \beta_6$	$\alpha_6 \cdot \beta_6$	$\alpha_5 \cdot \beta_6$	$\alpha_4 \cdot \beta_6$	$\alpha_3 \cdot \beta_6$	$\alpha_2 \cdot \beta_6$	$\alpha_1 \cdot \beta_6$	$\alpha_0 \cdot \beta_6$						
		$\alpha_7 \cdot \beta_7$	$\alpha_6 \cdot \beta_7$	$\alpha_5 \cdot \beta_7$	$\alpha_4 \cdot \beta_7$	$\alpha_3 \cdot \beta_7$	$\alpha_2 \cdot \beta_7$	$\alpha_1 \cdot \beta_7$	$\alpha_0 \cdot \beta_7$							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Figure 2-4 Example of partial product matrix of an unsigned 8×8 multiplier.

Partial products belonging to one column from top to bottom are donated as: $p_1, p_2, \ldots, p_m$, and m is the number of partial products in this column. For example, as shown in Figure 2-5, partial products in the column of bit 5 are denoted as $p_1, p_2, \ldots, p_6$, where $p_1 = \alpha_5 \cdot \beta_0$, $p_2 = \alpha_4 \cdot \beta_1, \ldots, p_6 = \alpha_0 \cdot \beta_5$.

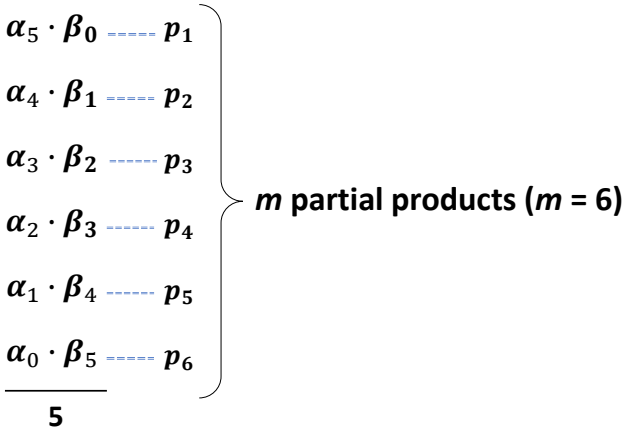


Figure 2-5 Example of partial products in the column of bit 5.

The arithmetic sum result of m partial products belonging to the same column, R^m, is described as:

$$R^m = \sum(p_1, p_2, \dots, p_m). \tag{2-3}$$

where notation Σ means the summation of these elements. R^m ranges from 0 to m. Assume that two inputs of the multiplier are uniformly and independently distributed, hence the probability is 0.75 and 0.25 for the cases that one partial product is equal to '0' and '1', respectively.

In a multiplier, compressors and adders are usually used to accumulate partial products. The inputs of compressor are all partial products belonging to one column. The probability distribution of the arithmetic sum results of m partial products in an 8×8 multiplier is illustrated in Figure 2-6.

Consider that different cases that the number of partial products, that is m, ranges from 2 to 8. For all cases, the probability concentrates on the range from 0 to 2 of R^m, and decreases on the range from 2 to m.

Furthermore, the probability distribution on the multiplications in image sharpening filter on LENA image is shown in Figure 2-7. The tendency in Figure 2-7 is a bit different with that in Figure 2-6, but the probability also concentrates on the range from 0 to 2 of R^m. The probability is close to 0 for the range of R^m is larger than 2.

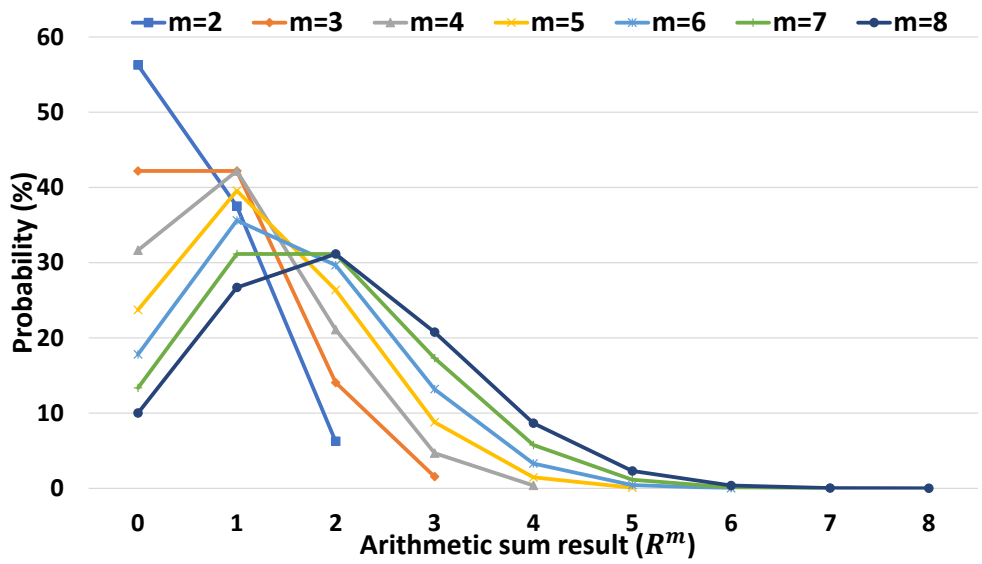


Figure 2-6 Probability distribution of the arithmetic sum result of m partial products in an 8×8 multiplier.

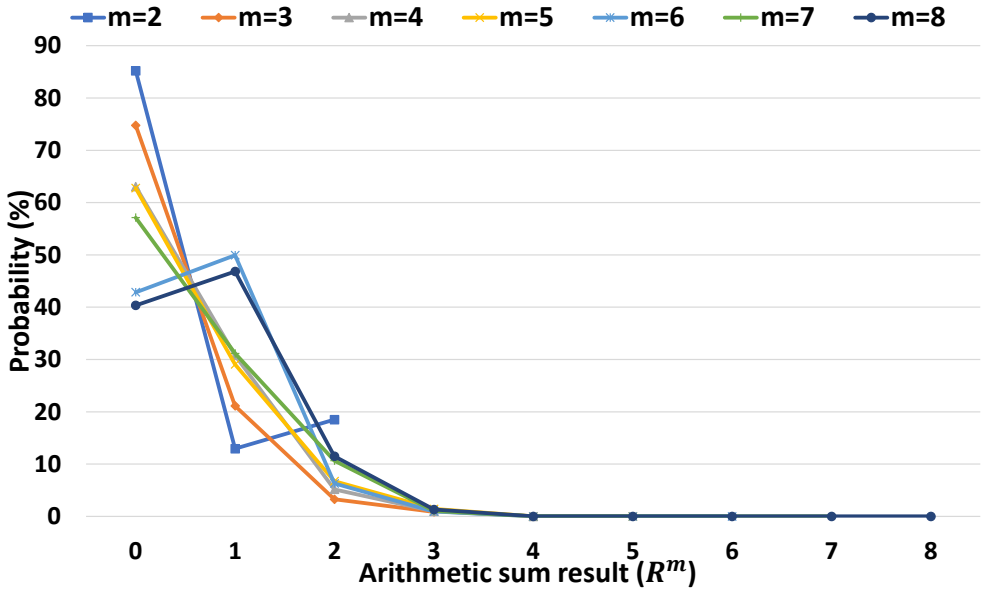


Figure 2-7 Probability distribution of the arithmetic sum result of m partial products in an image processing.

2.2.4 Design of Probability-Driven Inexact m:2 Compressor

According to the analysis in the last section, two primary observations are concluded:

i) the probability is low for the case that the sum result R^m is greater than 2;

ii) the probability of R^m on range 0~1 is higher than that on 2.

For inexact compressor design, based on the observation (i), two bits are sufficient to calculate the sum result of m partial products. Moreover, based on the observation (ii), one bit is sufficient for most situations. Therefore, one constraint is conducted based on the observation (ii), that is, if only one bit is equal to '1', then this '1' is distributed on the first bit (i.e. W_1^m in the following). By the constraint, an inexact half-adder with low error probability is proposed.

The design rule of the inexact m:2 compressor (m:2 Com) that calculates the arithmetic sum result of m partial products using two bits is given by

$$\widetilde{R^m} = \sum (W_1^m, W_2^m),$$
$$Constraint: W_1^m \geq W_2^m, \tag{2-4}$$

where $\widetilde{R^m}$ indicates the inexact sum result of m partial products. W_1^m and W_2^m have the same weights as m partial products. The value assignment of W_1^m and W_2^m are determined by R^m. When R^m is less than 2, one bit is sufficient to the sum result, hence W_1^m is equal to R^m. When R^m is equal to or greater than 2, both of two bits are '1'. The value assignment of W_1^m and W_2^m is given by

$$W_1^m = \begin{cases} 0 & R^m < 1 \\ 1 & R^m \geq 1 \end{cases} \tag{2-5}$$

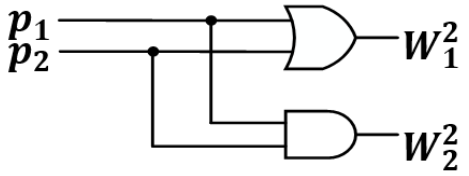
$$W_2^m = \begin{cases} 0 & R^m < 2 \\ 1 & R^m \geq 2 \end{cases} \tag{2-6}$$

According to Eq. 2-5 and 2-6, the logic functions of W_1^m and W_2^m can be deduced. In Eq. 2-5, R^m is equal to or greater than 1, that is, at least one of m partial products is '1'. Therefore, W_1^m can be calculated as the result of OR operation of all m partial products. For the value assignment of W_2^m in Eq. 2-6, R^m is equal to or greater than 2, that is, at least two partial products are '1'. Thus, W_2^m can be calculated as the result of AND operation of any two partial products. The logic functions of W_1^m and W_2^m can be expressed as:

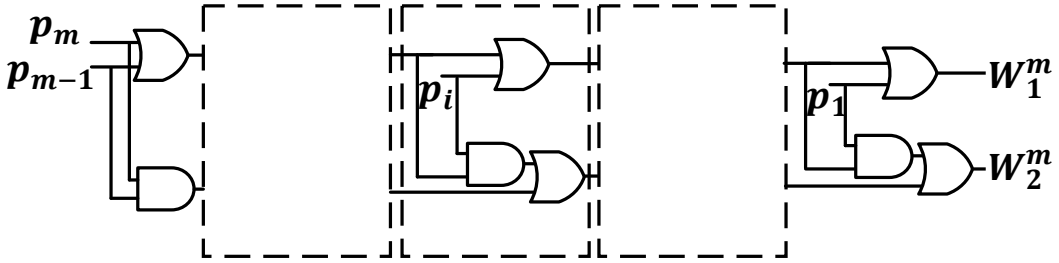
$$W_1^m = p_1 + p_2 + \cdots + p_m,$$
$$W_2^m = p_1 \cdot (p_2 + \cdots + p_m) + \cdots + p_i \cdot (p_{i+1} + \cdots + p_m) + \cdots + p_{m-1} \cdot p_m, \tag{2-7}$$

where '+' indicates OR operation and '·' means AND operation. In the functions of

W_1^m and W_2^m, no XOR gates are required, hence we foresee the possibility of simpler circuit. Figure 2-8 shows the structure for m:2 Com. When m is equal to 2, the circuit includes one AND gate and one OR gate. When m is greater than 2-input, there is a common architecture as shown in Figure 2-8(b).



(a) 2:2 Com.



(b) m:2 Com (m>=3). Dotted box indicates the reduplicative operation.

Figure 2-8 Architecture of the m:2 Com.

When m is equal to 2, the compressor can be regarded as inexact half-adder (inHA). In the following, inHA, 4:2 compressor (4:2 Com) ,6:2 compressor (6:2 Com) and 8:2 compressor (8:2 Com) are discussed in detail, because these four are employed in the proposed approximate multiplier.

1) Inexact half-adder (inHA)

Consider two partial products belonging to the same column, the exact sum result R^2 is calculated as:

$$R^2 = \sum(p_1, p_2). \tag{2-8}$$

The maximum value of R^2 is 2. With the help of Eq. 2-4, 2-5 and 2-6, the behavior of inHA can be obtained, as shown in Table 2-1.

Table 2-1 The behavior of ineact half-adder (inHA).

inputs		exact result	inHA			probability
p_1	p_2	R^2	W_1^2	W_2^2	$\widetilde{R^2}$	
0	0	0	0	0	0	0.5625
0	1	1	1	0	1	0.1875
1	0	1	1	0	1	0.1875
1	1	2	1	1	2	0.0625

As illustrated in Table 2-1, the first output W_1^2 of inHA is equal to the exact result R^2, for three input combinations ('00', '01' and '10'), with the occurrence probability of 0.9375. Thus, W_1^2 is sufficient for these combinations. In addition, $\widetilde{R^2}$ of inHA is always equal to the exact result R^2 for all combinations. It is a feature of inHA that W_2^2 can be regarded as an error compensation bit to sum with W_1^2 as error recovery.

With the help of Eq. 2-7, the function of inHA is given by

$$W_1^2 = p_1 + p_2,$$
$$W_2^2 = p_1 \cdot p_2. \tag{2-9}$$

Compared with the conventional half-adder, the functions of two outputs in inHA are simpler. One XOR gate in conventional half-adder is replaced by one OR gate in the proposed inHA.

2) Inexact 4:2 compressor (4:2 Com)

The exact sum result of four partial products is calculated as:

$$R^4 = \sum(p_1, p_2, p_3, p_4). \tag{2-10}$$

R^4 ranges from 0 to 4. The behavior of 4:2 Com is shown in Table 2-2, with the help of Eq. 2-4, 2-5 and 2-6. The combination that R^4 is equal to 3 occurs with the probability of 0.0117. In addition, another input combination, '1111', occurs with the probability of 0.0039. Thus, the total probability that R^4 is greater than 2 is 0.0507. It indicates the error probability, which is the occurrence probability of difference between the exact result and inexact result.

The function of 4:2 Com can be written as:

Table 2-2 The behavior of ineact 4:2 compressor (4:2 Com).

inputs				exact result	4:2 Com			probability
p_1	p_2	p_3	p_4	R^4	W_1^4	W_2^4	$\widetilde{R^4}$	
0	0	0	0	0	0	0	0	0.3164
0	0	0	1	1	1	0	1	0.1055
0	0	1	0	1	1	0	1	0.1055
0	0	1	1	2	1	1	2	0.0352
0	1	0	0	1	1	0	1	0.1055
0	1	0	1	2	1	1	2	0.0352
0	1	1	0	2	1	1	2	0.0352
0	1	1	1	3	1	1	2	0.0117
1	0	0	0	1	1	0	1	0.1055
1	0	0	1	2	1	1	2	0.0352
1	0	1	0	2	1	1	2	0.0352
1	0	1	1	3	1	1	2	0.0117
1	1	0	0	2	1	1	2	0.0352
1	1	0	1	3	1	1	2	0.0117
1	1	1	0	3	1	1	2	0.0117
1	1	1	1	4	1	1	2	0.0039

* Gray background indicates error occurrence.

$$W_1^4 = p_1 + p_2 + p_3 + p_4,$$
$$W_2^4 = p_1 \cdot (p_2 + p_3 + p_4) + p_2 \cdot (p_3 + p_4) + p_3 \cdot p_4. \quad (2\text{-}11)$$

Equation 2-11 shows the possibility of simplifying the circuit complexity due to no XOR gates in 4:2 Com.

3) Inexact 6:2 compressor (6:2 Com)

Similar as the inHA and 4:2 Com, the function of 6:2 Com can be conducted based on the Eq. 2-7. Firstly, the exact sum result (i.e. R^6) of six partial products can be expressed as

$$R^6 = \sum(p_1, p_2, p_3, p_4, p_5, p_6). \quad (2\text{-}12)$$

Table 2-3 illustrates the behavior of 6:2 Com. Because the space limitation, input combinations are not included in this table. The term of exact result indicates the corresponding combinations. R^6 ranges from 0 to 6. Error occurs when the exact

results is greater than 2, which means there are more than two '1' in the input combinations. The error probability of 6:2 Com is 0.1694. The function of 6:2 Com is given as:

$$W_1^6 = p_1 + p_2 + p_3 + p_4 + p_5 + p_6,$$
$$W_2^6 = p_1 \cdot (p_2 + p_3 + p_4 + p_5 + p_6) + p_2 \cdot (p_3 + p_4 + p_5 + p_6) + p_3 \cdot (p_4 + p_5 + p_6) + p_4 \cdot (p_5 + p_6) + p_5 \cdot p_6. \quad (2\text{-}13)$$

Table 2-3 The behavior of ineact 6:2 compressor (6:2 Com).

Exact result	**6:2 Com**			**probability**
$\boldsymbol{R^6}$	$\boldsymbol{W_1^6}$	$\boldsymbol{W_2^6}$	$\boldsymbol{\widetilde{R^6}}$	
0	0	0	0	0.1780
1	1	0	1	0.3560
2	1	1	2	0.2966
3	1	1	2	0.1318
4	1	1	2	0.0330
5	1	1	2	0.0043
6	1	1	2	0.0003

* Gray background indicates error occurrence.

Table 2-4 The behavior of ineact 8:2 compressor (8:2 Com).

Exact result	**8:2 Com**			**probability**
$\boldsymbol{R^8}$	$\boldsymbol{W_1^8}$	$\boldsymbol{W_2^8}$	$\boldsymbol{\widetilde{R^8}}$	
0	0	0	0	0.1001
1	1	0	1	0.2670
2	1	1	2	0.3115
3	1	1	2	0.2076
4	1	1	2	0.0865
5	1	1	2	0.0231
6	1	1	2	0.0038
7	1	1	2	0.0004
8	1	1	2	0.00002

* Gray background indicates error occurrence.

4) Inexact 8:2 compressor (8:2 Com)

8:2 Com has the similar design as inHA, 4:2 Com and 6:2 Com. Using Eq. 2-3, we first consider the exact sum result (i.e. R^8) of eight partial products. The range of R^8 is 0~8. Using the proposed approach presented in Eq. 2-4, 2-5 and 2-6, 8:2 Com is explored.

The behavior of 8:2 Com is shown in Table 2-4. In 8:2 Com, error occurs when R^8 is greater than 2, with a probability of 0.3214. The dominant part of the error probability is from combinations that R^8 is equal to 3, which is up to 0.2076. However, the error distance for this combination is small as 1 (= |3 – 2|). More importantly, the utilization of 8:2 Com is limited in the proposed multiplier, which also decreases the impact of error from 8:2 Com.

2.3 Approximate Multiplier Design using Proposed Inexact Compressor

The approximate multiplier design is proposed by utilizing proposed inexact compressors. This approximate multiplier includes three steps. Firstly, one multiplier is divided into three blocks with architectural-space construction. Then, results from three blocks are accumulated in parallel. Finally, a grouped error recovery scheme is explored to improve accuracy.

2.3.1 An 8×8 Multiplier with Different Approximation Levels on 4×4 Multipliers

An 8×8 multiplier can be constructed from four 4×4 multipliers following Eq. 2-1. Four 4×4 multipliers are classified into three multiplication blocks (High_block, Low_block and Mid_block), according to their significances.

i) *High_block*: This block contains $A_H \times B_H$ which is the most important. A Wallace tree is used to compute the resulting row (named as SH).

iii) *Low_block*: This block involves $A_L \times B_L$ and is the least important. OR gates are used to compute the result (named as SL) of this block.

iv) *Mid_block*: Figure 2-9 shows Mid_block containing $A_L \times B_H$ and $A_H \times B_L$. Two 4:2 Coms are used on bits 1 and 5, along with two 6:2 Coms are employed on bits 2 and 4. Elements of bit 3 are accumulated by one 8:2 Com. Eight rows are reduced to two rows (named as $SM1$ and $SM2$) using the proposed inexact compressors.

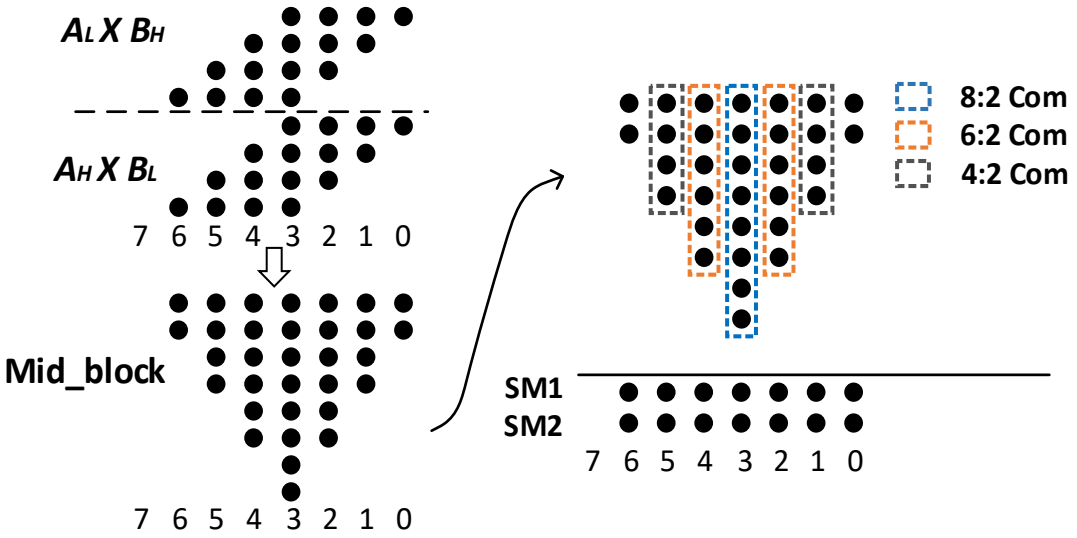


Figure 2-9 Dot notation for Mid_Block, where the proposed inexact compressors are used in partial product accumulation.

2.3.2 Accumulating Results of Three Blocks in Parallel

Figure 2-10 shows the overall structure of an approximate 8×8 multiplier. Note that the partial product generation and error-recovery scheme are not shown.

In Stage 1, an 8×8 multiplier is divided into three blocks with different precision operations, as described in Section 2.3.1. The results are SH, SL, $SM1$ and $SM2$.

In Stage 2, three rows from three blocks are reduced into two rows using carry save adder (CSA) without carry propagation. Six full-adders (FAs) are used on the bits 4 to 10, except the bit 7. One half-adder (HA) is required for the bit 7. The results of this stage are named as $S3$ and C.

In Stage 3, $S3$ and C are processed by the proposed inHAs. According to the feature of inHA, the second output W_2^2 of an inHA can be regarded as error compensation bit. The results of this stage are one sum result row (named as S) and one error compensation row (named as E).

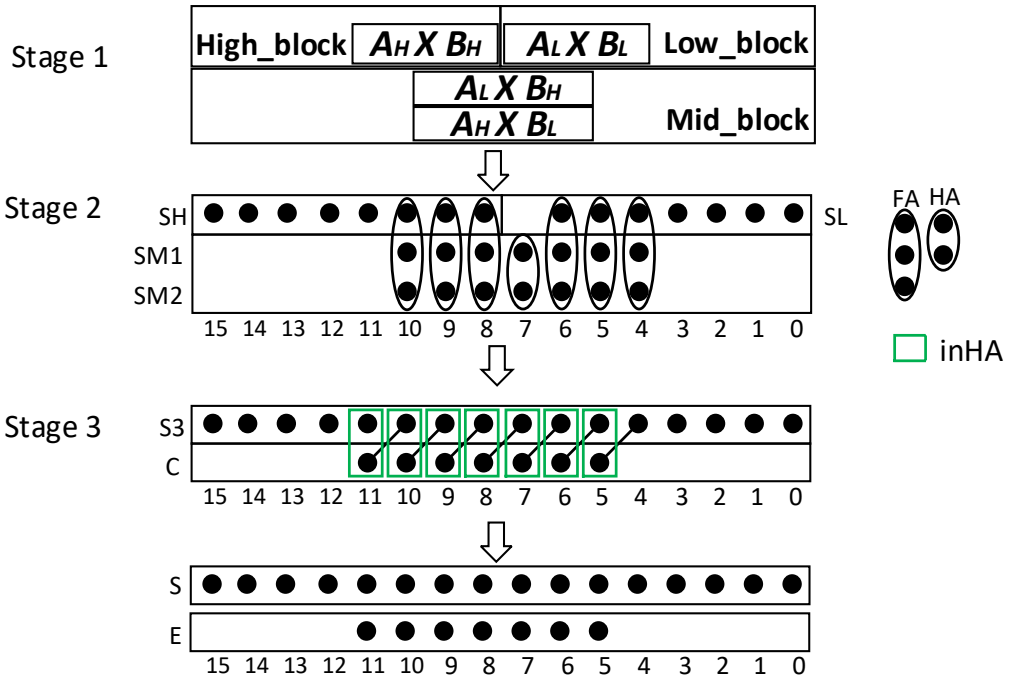
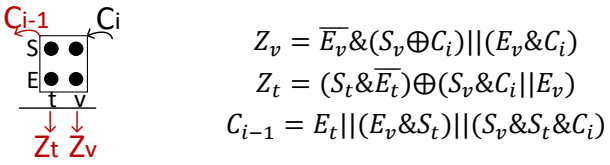


Figure 2-10 Overall structure of an 8×8 multiplier.

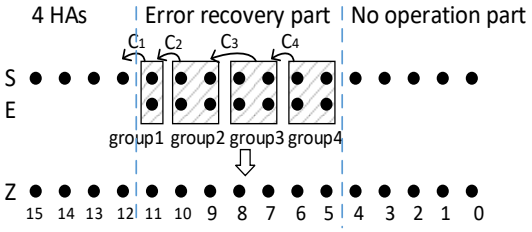
2.3.3 A Grouped Error Recovery Scheme

To improve accuracy, a grouped error recovery scheme is proposed, which is explored based on the feature of the proposed inHA.

S and E in Stage 3 are produced by inHAs. S must be '1' when E is '1' (determined by Equation 2-9). Thus, the function of conventional adder for summing S and E can be simplified. This error recovery scheme can be implemented by simplified adders to add E into S. As shown in Figure 2-11, seven E bits are divided into four groups. The carry results between two adjacent groups are produced in serial, while the sum results in one group are produced in parallel. The function of carry result is simplified, hence the critical path is shortened.



(a)



(b)

Figure 2-11 A grouped error recovery scheme. (a) The function of each group. (b) Overall structure.

Table 2-5 Five variants of error recovery.

Designs	**Group numbers of error compensation bits**	**Corresponding bit-positions**
MGER-0g	No error recovery	Null
MGER-1g	group1	11
MGER-2g	group1 group2	9-11
MGER-3g	group1 group2 group3	7-11
MGER-4g	group1 group2 group3 group4	5-11

Table 2-5 illustrates all variants of error recovery. A variant of the approximate multiplier with grouped error recovery is referred as MGER-kg, where k is the number of groups of error compensation elements. For example, MGER-3g means that three error compensation groups (bits 7 to 11 in E) are used to compensate error. Four accurate HAs generate the values from bits 12 to 15, along with the values of bits 0 to 6 are kept as these bits in S.

2.4 Performance Evaluation

In this section, the impact of probability-driven inexact compressors is firstly evaluated. Then, the accuracy evaluation of approximate multipliers is presented,

followed by the hardware theoretical analysis and synthesized results. The proposed multipliers (MGER-0g, MGER-1g, …, MGER-4g) were compared with the Wallace multiplier and approximate multipliers proposed in [52] (ATCM1, ATCM2), [50] (MUL1), [39] (AMLC2) and [45] (AM1-t). In [39], the value 2 for AMLC2 means that the 2-cluster logic compression is used to accumulate partial products. In [45], the parameter t for AM1-t is the number of MSBs for error recovery.

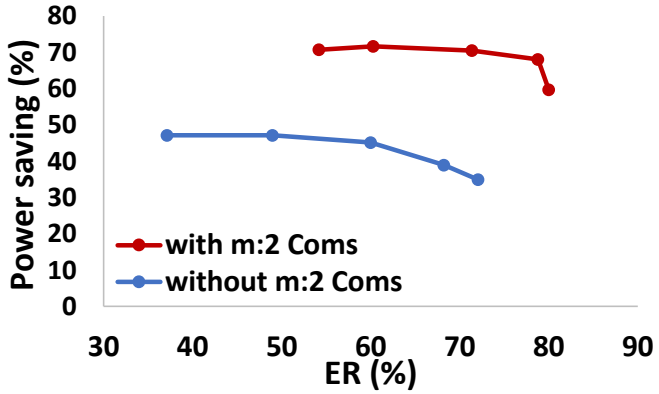
2.4.1 Evaluation for the Impact of Inexact Compressors

a) Comparison of proposed multipliers with/without proposed compressors

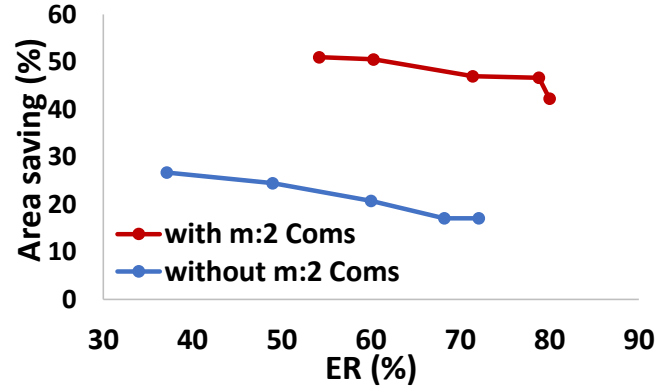
In order to evaluate the effectiveness of proposed inexact compressors in the multiplier, proposed approximate multipliers with/without m:2 Coms (the proposed compressors) are implemented. Approximate multipliers with m:2 Coms are the design of MGER-kg presented in Section 2.3. Approximate multipliers without m:2 Coms mean that operation using m:2 Coms is replaced with exact full-adders and half-adders while other operations are unchanged.

Approximate multipliers with different cases have been designed in Verilog and synthesized using the Synopsys Design Compiler with SMIC 40nm process library. The exact multiplier has been synthesized and evaluated in the same environment. The operating condition for synthesis was typical condition, where the process factor is 1.00, the power supply is 1.1v, and the operating temperature is 25℃. All multipliers were synthesized and optimized with default compile options. For power evaluation, the power consumption was evaluated at 0.5GHz frequency using the Synopsys Power Compiler with a switching activity interchange format file generated from all input combinations of 65536 cases for 8×8 multipliers. Error rate (ER) is the percentage of the erroneous result produced by approximate multipliers among all results, which shows the accuracy loss compared with the exact multiplier.

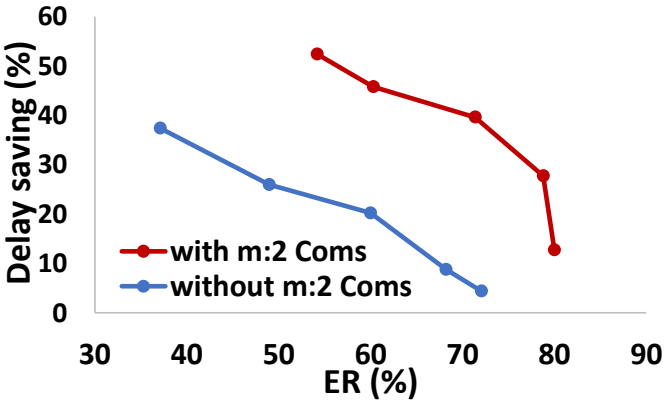
Figure 2-12 shows the savings of power, area and delay achieved by the approximate multipliers, compared with the exact multiplier. Red line shows the evaluation of multipliers with m:2 Coms and blue line shows the approximate



(a)



(b)



(c)

Figure 2-12 Hardware saving versus error rate for the approximate multipliers with/without proposed probability-driven inexact m:2 Coms. (a) Power saving vs. ER. (b) Area saving vs. ER. (c) Delay saving vs. ER.

multipliers without m:2 Coms. Approximate multipliers with m:2 Coms incur more than 11.66% accuracy loss on average, compared with those without m:2 Coms. However, in all evaluations, red lines are above blue lines, which shows the multipliers

employing m:2 Coms achieves more savings than those without m:2 Coms. For example, approximate multipliers without m:2 Coms achieves area saving of 21.20% on average, while the area saving by approximate multipliers with m:2 Coms is up to 47.48%. More hardware savings can be achieved by using m:2 Coms, while the accuracy loss is acceptable. This shows the effectiveness of the proposed m:2 Coms in the multiplier.

b) Comparison of the proposed compressors with previous approximate compressors

In this subsection, a hardware comparison of proposed compressors and previous inexact compressors is presented to further show the advantage of the proposed inexact compressor design. The proposed design was implemented and evaluated at the condition reported in Section 2.4.1 a). All previous compressors were implemented based on the logic function presented in their papers. The proposed compressors and previous compressors were evaluated and compared under the same condition. Table 2-6 shows the synthesized results.

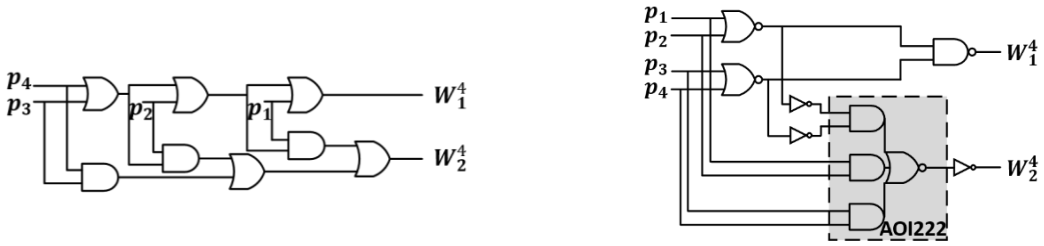
Table 2-6 Comparison of synthesized results for inexact compressors.

Designs	**Power (uW)**	**Area (um^2)**	**Delay (ns)**
2-input compressor in proposed design	0.43	3.19	0.09
4-input compressor in proposed design	0.36	7.34	0.32
6-input compressor in proposed design	0.46	12.45	0.54
8-input compressor in proposed design	0.38	15.64	0.46
2-input compressor in [39]	0.08	1.28	0.05
4-input compressor in [45]	0.55	6.38	0.15
2-input compressor in [50]	0.43	3.19	0.09
3-input compressor in [50]	0.66	4.79	0.16
4-input compressor in [50]	0.51	9.58	0.17
2-input compressor in [52]	0.43	3.19	0.09
4-input compressor in [52]	1.13	12.45	0.40

For inexact compressors, the hardware performance of power and area are more important than delay. There are two reasons: firstly, the partial product accumulation usually includes exact compressors and inexact compressors. The delay in this step is

determined by the exact compressor, rather than inexact compressor. The second reason is the primary reason, which the major delay of one multiplier is caused by the final carry propagate adder. Thus, the delay of inexact compressor has minor effect to the delay of the overall multiplier.

Therefore, here the discussion on power and area are stated. Regarding the power consumption shown in Table 2-6, the proposed design has less power consumption than previous compressors. More importantly, the proposed design first extends the input-size to 6- and 8-bit. It can be observed that the proposed large compressor even has less power consumption than 4-bit compressors [45, 50, 52]. As for area, the proposed 6- and 8-input compressors have the large area. However, accumulating the fixed elements with the small compressor requires more compressors. For example, there needs two 4-input compressors to accumulate 8 elements, which roughly costs 19.16 um^2 for [50], and 24.9 um^2 for [52]. In [45], the 4-input compressor actually processes two bits at the current column and two bits at the preceding column (as the predict carry). Therefore, to accumulate 8 elements at the same column, it requires four compressors and area is 25.52 um^2. In contrast, accumulating 8 elements with the proposed 8-input compressor requires 15.42 um^2. Therefore, the proposed compressors cost smaller area than previous compressors to accumulate the fixed number of elements.



(a) (b)

Figure 2-13 4:2 Compressor (4:2 Com). (a) Logical AND-OR structure. (b) Cell structure after synthesis.

To explain this superiority, feature of the proposed design is explored. The power and area savings by the proposed design are mainly due to the feature that compressors only comprise AND and OR gates. During the synthesis, AND and OR gates can be

synthesized to compound gate, which has less transistors as a power- and area-efficient cell. To demonstrate this feature, the logic circuit based on function and schematic circuit from synthesis are illustrated. Take the proposed 4:2 Com as an example to explain, Figure 2-13 (a) shows the logical AND-OR circuit drawn from the function of 4:2 Com as stated in Eq. 2-11 (Section 2.2.4). The schematic circuit extracted after the synthesis is shown in Figure 2-13 (b). It can be observed that the synthesized inexact compressor is simpler than original circuit due to the compound gates. In this example, the compound gate AOI222 is included, which has less transistors.

2.4.2 Accuracy Evaluation

a) Metrics for accuracy analysis

Here gives several metrics to evaluate the accuracy of approximate multipliers. For multiplier, the error distance (ED) is defined as the arithmetic absolute difference between the exact product (Y) and the inexact product ($\tilde{Y}$), that is,

$$ED = |\tilde{Y} - Y|. \tag{2-14}$$

MED is the average value of EDs for a set of outputs. The normalized mean error distance (NMED) is developed in [57], which is applicable to compare multipliers with different sizes, as defined as

$$NMED = MED/Y_{max}. \tag{2-15}$$

where Y_{max} is the maximum output of an exact multiplier, that is, $(2^n - 1)^2$ for an $n \times n$ multiplier. The relative error distance (RED) is calculated as

$$RED = ED/Y. \tag{2-16}$$

The average value of REDs is mean RED (MRED).

In addition, normalized worst case error distance (NWCE) and worst case relative error (WCRE) also are important metrics. This is because low mean error but excessively high ED in some cases also could cause the unacceptable results. NWCE is defined as the normalized value of the maximum ED, expressed as

$$NWCE = ED_{max}/Y_{max}. \tag{2-17}$$

WCRE is the maximum value of RED among all results. The functional models for

approximate multipliers were implemented using Matlab. An exhaustive simulation was performed for each 8×8 multiplier (65536 patterns).

b) Accuracy analysis

Table 2-7 illustrates the NMED, MRED, ER, NWCE and NWCE results for the existing approximate multipliers and the proposed multiplier. For the proposed multiplier design, error varies with error recovery configurations. Error decreases drastically from MGER-1g to MGER-2g. For example, the MRED decreases from 6.19% (MGER-1g) to 2.87% (MGER-2g). The reason is that the additional error compensation part is group 2, whose corresponding weights are 2^9 and 2^{10}. These weights are significant for overall multiplication result. Regarding the mean error metrics (NMED, MRED and ER), error of AM1-10 is the lowest. In terms of NMED, ATCM2 also achieves the lowest value as 0.21%. The mean error of the proposed MGER-4g is slightly higher than that of AM1-10.

Table 2-7 Accuracy comparisons for approximate multipliers.

Designs	NMED (%)	MRED (%)	ER (%)	NWCE (%)	WCRE (%)
MGER-0g	1.30	7.86	80.00	8.81	56.58
MGER-1g	0.94	6.19	78.79	6.40	56.58
MGER-2g	0.40	2.87	71.38	5.51	56.58
MGER-3g	0.28	1.44	60.26	5.12	49.46
MGER-4g	0.25	1.07	54.18	**4.97**	43.56
ATCM1 [52]	0.28	1.64	55.44	11.92	44.44
ATCM2 [52]	**0.21**	1.21	47.33	11.23	44.44
MUL1 [50]	2.58	7.86	81.79	27.60	33.86
AMLC2 [39]	0.35	1.99	49.11	5.64	**33.20**
AM1-4 [45]	1.85	10.32	81.06	13.63	61.36
AM1-6 [45]	0.73	5.08	76.41	12.06	59.59
AM1-8 [45]	0.30	1.87	61.55	10.88	57.44
AM1-10 [45]	**0.21**	**0.79**	**40.97**	10.78	57.14

In terms of worst case, MGER-4g has smaller NWCE than ATCM2 and AM1-10, whose mean errors are smaller than MGER-4g. In the proposed design, the High_block

is fully accurate, hence this part ensures the low error in columns with large weights. In addition, the approximation in MGER-4g is applied on columns from 2^0 to 2^{10}. However, the approximate compression is used on columns from 2^0 to 2^{13} in ATCM2, and all columns in AM1-10, which incurs the error in columns with large weights. On the other hand, the maximum weight of bit difference mainly causes the maximum ED. We further explored the input combination for the maximum ED, where (255×255) are both for ATCM2 and MGER-4g, and (238×255) is for AM1-10. The maximum weight of bit difference is 2^{12} in ATCM2, yet 2^{11} in MGER-4g. The compression logic in AM1-10 considers the lower one bit, hence the input operands for worst case is less than 255. The maximum weight of bit difference is 2^{13} in AM1-10. In terms of WCRE, the value of MGER-4g ranks third among all approximate multipliers. In general, the maximum RED occurs in the multiplication of two small numbers, such as (15×15) in MGER-4g. The higher degrees of approximation in low part cause the larger WCRE. In the proposed design, all bits in lower half part are accumulated approximately, while some bits in MUL1 and AMLC2 keep accurate. Therefore, AMLC2 and MUL1 have the lower WCREs than MGER-4g. The mean error combined with the error in the worst case indicates the comprehensive accuracy loss. On the whole, MGER-4g achieves the better performance on accuracy than other designs.

2.4.3 Hardware Analysis

a) Area and Delay Estimation

Here shows the evaluation of area and delay by the number of gates. Consider a 2-input gate (e.g. AND gate) as a unit. For one unit, the area is α_0 and the delay is τ_0. The 2-input XOR gate is regarded as two units, with $2\alpha_0$ area and $2\tau_0$ delay. The area for FA and HA is $7\alpha_0$ and $3\alpha_0$, and the delay for FA and HA is $4\tau_0$ and $2\tau_0$, respectively [45][52]. Note that the carry propagation delay of FA and HA in an adder chain is $2\tau_0$ and τ_0, respectively.

For area estimation, all operations are converted to units and count the total number

of units. The partial product accumulation of MGER-kg includes three operations on three blocks (i.e. Stage 1), CSA stage (i.e. Stage 2) and inHA stage (i.e. Stage 3). In Stage 1, High_block consists of nine FAs and three HAs which are converted into 72 units, along with nine OR gates in Low_block. Mid_block involves 16 units for two 4:2 Coms, 28 units for two 6:2 Coms, and 20 units for one 8:2 Com. Therefore, the number of units of High_block, Mid_block and Low_block is 72, 64 and 9, respectively. The total area of Stage 1 is $145\alpha_0$. In Stage 2, six FAs and one HA are converted to 45 units, hence the area is $45\alpha_0$. Stage 3 involves seven inHAs with the area of $14\alpha_0$. Therefore, the area of partial product accumulation step is $204\alpha_0$.

The cost of error recovery step in MGER-kg is determined by the number of error compensation groups. The area of MGER-kg is the total area of both partial product accumulation step and error recovery step, i.e.,

$$A_{MGER-kg} = 204\alpha_0 + \alpha_{kg}, \tag{2-18}$$

where α_{kg} is the area of error recovery step in MGER-kg.

For delay estimation, the critical path of MGER-kg is throughout the partial product accumulation step and error recovery step. Note that, only lower three LSBs in High_block are used by CSAs in Stage 2. Thus, the critical path of Stage 1 is determined by the 8:2 Com in Mid_block, which is $13\tau_0$. For Stage 2 and Stage 3, the delay is $4\tau_0$ (FA) and τ_0 (inHA), respectively. The delay of partial product accumulation step is $18\tau_0$. The total delay of MGER-kg is calculated as:

$$D_{MGER-kg} = 18\tau_0 + \tau_{kg}, \tag{2-19}$$

where τ_{kg} is the delay of error recovery step in MGER-kg.

In an 8×8 Wallace multiplier, four CSA stages including 38 FAs and 15 HAs compresses eight rows of partial products into two rows. These adders are converted to 311 units and the delay is $16\tau_0$. The final CPA involving 10 FAs and 1 HAs is converted into 73 units and the area is $73\alpha_0$. The delay of this CPA is determined by the carry propagation chain, thereby the delay is $22\tau_0$.

Table 2-8 shows the estimated area and delay of the Wallace multiplier and the proposed multipliers. The dominated delay consumption of the proposed multipliers is determined by partial product accumulation step. In this step, the proposed multipliers

have larger estimated delay than the Wallace multiplier. However, the circuit complexity of the error recovery step is much simpler than that of conventional CPA. Therefore, the proposed multipliers have less total delay than the Wallace multiplier.

Table 2-8 Area and delay esitimation.

Designs	Partial product accumulation		Final addition		Total	
	Area (α_0)	Delay (τ_0)	Area (α_0)	Delay (τ_0)	Area (α_0)	Delay (τ_0)
Wallace	311	16	73	22	384	38
MGER-0g	204	18	0	0	204	18
MGER-1g	204	18	13	5	217	23
MGER-2g	204	18	25	9	229	27
MGER-3g	204	18	40	12	244	30
MGER-4g	204	18	55	15	259	33

Table 2-9 Synthesized results comparison.

Designs	Area (um^2)	Power (uW)	Delay (ns)	PDP (fJ)	ADP ($um^2 \cdot ns$)
Wallace	496.04	92.03	2.27	208.91	1126.01
MGER-0g	**243.23**	26.92	**1.08**	**29.07**	**262.69**
MGER-1g	245.46	**26.07**	1.23	32.07	301.92
MGER-2g	263.02	27.16	1.37	37.21	360.34
MGER-3g	264.62	29.40	1.64	48.22	433.98
MGER-4g	285.36	37.04	1.98	73.34	565.01
ATCM1 [52]	281.53	44.86	1.71	76.71	481.42
ATCM2 [52]	314.73	49.00	1.72	84.28	541.34
MUL1 [50]	261.74	48.14	1.77	85.21	463.28
AMLC2 [39]	277.07	45.10	1.71	77.12	473.79
AM1-4 [45]	267.81	38.56	1.25	48.20	334.76
AM1-6 [45]	301.96	40.37	1.61	65.00	486.16
AM1-8 [45]	344.42	49.71	1.99	98.92	685.40
AM1-10 [45]	385.27	61.39	2.42	148.56	932.35

b) Synthesis Results

The proposed multipliers (MGER-kg) have been simulated and synthesized under the same conditions as Section 2.4.1. Existing approximate multipliers and the Wallace multiplier (exact) were implemented based on algorithms in their papers, which were

synthesized and evaluated in the same environment as the proposed design. All designs were synthesized without the timing and area constraints. The condition for the power evaluation was the same as Section 2.4.1. Table 2-9 shows the synthesized results for power consumption, circuit area, critical path delay, power-delay product (PDP) and area-delay product (ADP).

Compared with the Wallace multiplier, the proposed multipliers deliver power reduction of 59.75%~70.75%. The area and delay reduction is up to 50.97% and 52.42%, respectively. Compared with the theoretical results shown in Table 2-7, the hardware reduction after synthesis is larger than the estimated result. Take MGER-3g as an example, for the area (delay) reduction, the experimental result is 46.65% (27.75%), while the estimated result is 36.46% (21.05%).

Here are two primary reasons might cause these differences, which were found from netlist and critical path reports. Firstly, the majority of gates in the Wallace multiplier are XOR gates, which is 91 2-input XOR gates in partial product accumulation. During DC synthesis, XOR gates are difficult to be synthesized to compound gates. In contrast, in the proposed design, the majority of gates in partial product accumulation are AND and OR gates, which can be synthesized to compound gates, such as OAI222. Secondly, in terms of critical path, the Wallace multiplier in partial product accumulation involves 8 2-input XOR gates, while the proposed multiplier has 2 2-input XOR gates. Although the estimated delay for two multipliers is close, the proposed multiplier includes more basic gates (AND and OR gates), which can be synthesized to compound gates. It also demonstrates the validity of the proposed compressors, which does not include XOR gates.

Then, the number of XOR and XNOR gates of multipliers is shown to explain the advantage of the proposed compressors. The XOR and XNOR gates of each multiplier was obtained from the synthesized netlists shown in Table 2-10. The total number of XOR and XNOR gates of MGER-0g is the smallest among all multipliers. The number of XOR2 gate in MGER-0g is the smallest among all multipliers, and the number of XNOR2 gate is only 1. In terms of XNOR3 gate, the number in MGER-0g is greater than those in MUL1 and AM1-t. However, the number of XOR2 gate in those two

designs even four or fifteen times of that in MGER-0g.

Table 2-10 Number of XOR and XNOR gates.

Designs	XOR2	XNOR2	XNOR3	Total
Wallace	14	3	46	63
MGER-0g	2	1	12	15
MGER-1g	5	1	13	19
MGER-2g	6	1	14	21
MGER-3g	8	3	12	23
MGER-4g	10	3	11	24
ATCM1 [52]	5	0	16	21
ATCM2 [52]	8	1	20	29
MUL1 [50]	8	1	11	20
AMLC2 [39]	6	0	26	32
AM1-4 [45]	19	8	0	27
AM1-6 [45]	20	12	2	34
AM1-8 [45]	29	4	4	37
AM1-10 [45]	31	4	6	41

*XOR3 cell was not comprised in netlist.

The dynamic power usually accounts for a large percentage of the total power consumption in a combinational circuit. The dynamic power obtained by Power Compiler is the sum of switching power and internal power [58]. The simpler circuit leads to the less internal power. For the switching activity power (P_s), it is calculated as

$$P_s = \frac{Vdd^2}{2} \sum_{\forall net(i)} (C_{load_i} \times TR_i), \tag{2-20}$$

where C_{load_i} indicates the capacitive load of net i and TR_i means the toggle rate of net i. In general, the circuit complexity and switching activity of XNOR3 gate is larger than those of XNOR2/XOR2 gate. In addition, XNOR2/XOR2 gate has larger circuit complexity and switching activity than AND and OR gates. This is because that control signal for AND/OR gate is one input as ‘0’/‘1’, respectively. For example, when one input is ‘1’, the output of OR gate is determined as ‘1’. Hence, the switching transition on another input will not affect the output. However, there must be two control signals for 2-input XOR gate and three control signals for 3-input XOR gate. The switching

transition on one input of XOR gate usually affects a transition on the output. Therefore, the switching activity of XOR gate is higher than that of AND or OR gate. The small number of XOR gates in the proposed multiplier leads to low toggle rate. In addition, the proposed multipliers have the small capacitive load because of the small number of XOR gates. Therefore, the proposed multipliers have the lower power and PDP.

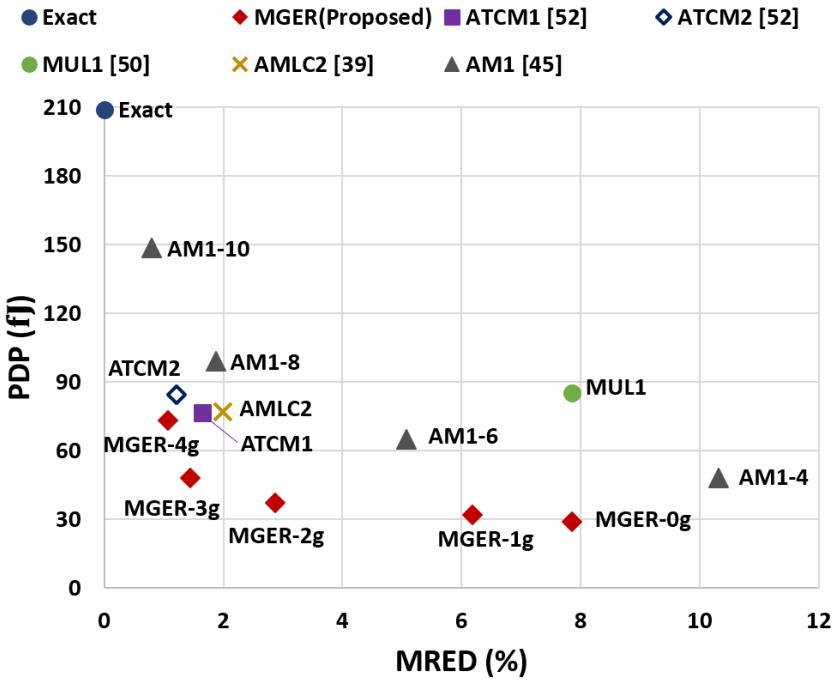


Figure 2-14 MRED and PDP for exact multiplier and approximate multipliers.

Figure 2-14 shows the overall comparison for different multiplier in terms of MRED and PDP. This is an intuitive comparison to show the trade-off between energy saving and accuracy loss. In general, MRED indicates the error distribution of approximate multipliers. In terms of the MRED-PDP, the proposed multipliers design achieves the better trade-off between MRED and PDP than other designs. Our design shows the smallest MRED on the same PDP, and the smallest PDP on the same MRED. It demonstrates that the proposed approximate multiplier design achieves the better balance than existing designs, and also achieves the target of this research.

2.4.4 Application of Approximate Multipliers to Image Processing

Approximate circuits can be used in error-tolerant applications. The image sharpening algorithm is widely used to evaluate approximate multipliers as shown in [59]. The image sharpening algorithm is given by

$$P(x,y) = 2I(x,y) - \frac{1}{273}\sum_{i=-2}^{2}\sum_{j=-2}^{2} G(i+3,j+3)I(x-i,y-j), \quad (2\text{-}21)$$

where G is a 5×5 matrix, given by

$$G = \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

In this application, G operates on 5×5 block in the image. I(x,y) indicates the original image, and P(x,y) is the processed image. The peak signal-to-noise ratio (PSNR) is a metric to measure the quality of processed image compared with the exact image, that is based on the mean squared error (MSE) [42], defined as:

$$MSE = \frac{1}{pq}\sum_{x=0}^{p-1}\sum_{y=0}^{q-1}[\hat{P}(x,y) - P(x,y)]^2, \quad (2\text{-}22)$$

$$PSNR = 10 \times log_{10}(\frac{{MAX_I}^2}{MSE}), \quad (2\text{-}23)$$

where P and $\hat{P}$ denotes the processed images using the exact multiplier and approximate multiplier, and p and q are the image dimensions. The input image for this application is 512×512 grayscale bitmap image with 8-bit pixels.

Another metric quantifies the difference between the processed images using the exact multiplier and approximate multiplier is structural similarity (SSIM). SSIM is a weighted combination of three comparison measurements [73] between P and $\hat{P}$ which is processed images using the exact multiplier and approximate multiplier, respectively. Three comparative measures are luminance (l), contract (c) and structure (s), which are calculated as:

$$l(P,\hat{P}) = \frac{2\mu_P\mu_{\hat{P}}+c_1}{\mu_P^2+\mu_{\hat{P}}^2+c_1} \quad (2\text{-}24)$$

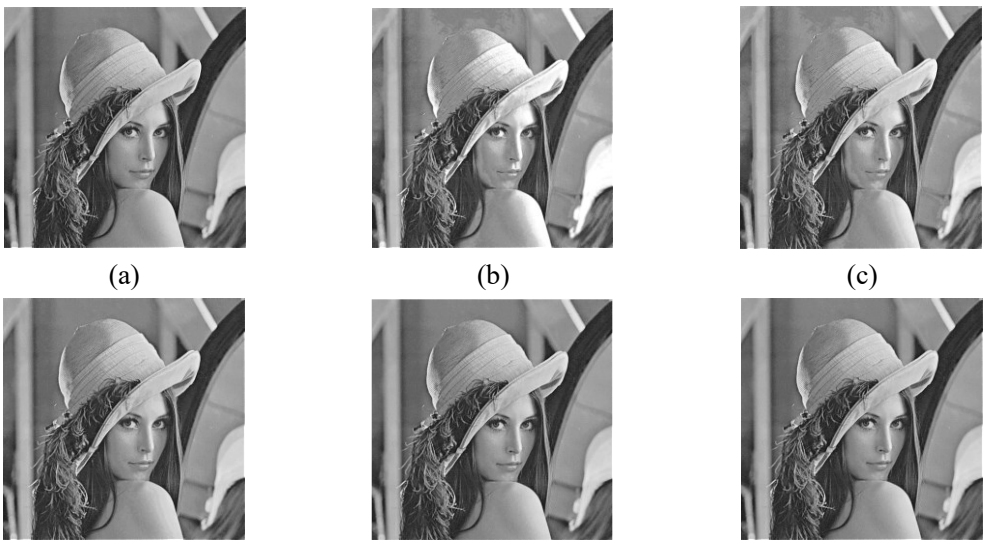
$$c(P,\hat{P}) = \frac{2\sigma_P\sigma_{\hat{P}}+c_2}{\sigma_P^2+\sigma_{\hat{P}}^2+c_2} \tag{2-25}$$

$$s(P,\hat{P}) = \frac{\sigma_{P\hat{P}}+c_3}{\sigma_P\sigma_{\hat{P}}+c_3} \tag{2-26}$$

where, μ_P and $\mu_{\hat{P}}$ is the average of image P and image $\hat{P}$, respectively. σ_P^2 and $\sigma_{\hat{P}}^2$ is the variance of image P and image $\hat{P}$, respectively, and $\sigma_P\sigma_{\hat{P}}$ is the covariance of images P and $\hat{P}$. In addition, c_1, c_2 and c_3 are constants to stabilize the division. For 8-bit pixels, c_1, c_2 and c_3 is set by default to 6.5025, 58.5525, and 29.2613, respectively. Then, the SSIM is the combination of those comparative measures as:

$$SSIM(P,\hat{P}) = [l(P,\hat{P}) \cdot c(P,\hat{P}) \cdot s(P,\hat{P})] \tag{2-27}$$

Figure 2-15 shows the processed images of exact multiplier and the proposed approximate multipliers. Table 2-10 illustrates the PSNR and SSIM values of processed images of all approximate multipliers.



(a) (b) (c)

(d) (e) (f)

Figure 2-15 Images processd by (a) Exact multiplier. (b) MGER-0g. (c) MGER-1g. (d) MGER-2g. (e) MGER-3g. (f) MGER-4g.

Table 2-11 PSNR and SSIM values for the image sharpening application.

Designs	PSNR (dB)	SSIM (%)
MGER-0g	23.40	94.09
MGER-1g	25.82	94.80
MGER-2g	34.65	99.07
MGER-3g	44.26	99.71
MGER-4g	48.29	99.81
ATCM1 [52]	49.48	99.85
ATCM2 [52]	52.37	99.85
MUL1 [50]	31.23	99.00
AMLC2 [39]	48.96	99.89
AM1-4 [45]	22.22	92.97
AM1-6 [45]	28.83	97.88
AM1-8 [45]	39.61	99.61
AM1-10 [45]	52.13	99.86

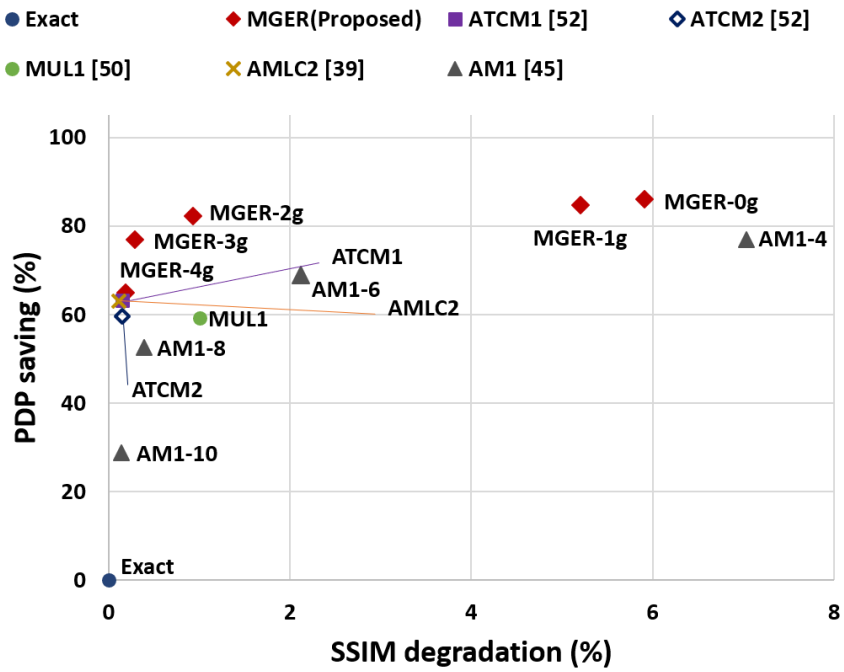


Figure 2-16 The trade-off between PDP saving and SSIM degradation for image sharpening application.

An intuitive comparison for all approximate multiplier with respect to the quality-power trade-off is shown in Figure 2-16. In this figure, x-axis reports the SSIM degradation, while y-axis reports the PDP saving delivered by the approximate

multiplier. We can observe that (i) for the range that SSIM reduction is less than 0.5%, MGER-3g achieves the highest PDP saving (76.92%); (ii) the proposed approximate multiplier design offers different configurations, which has a better quality-power trade-off than AM1.

2.5 Discussion on Extension to Signed Approximate Multiplier

Previous sections have presented the proposed ASIC-based approximates, mainly focusing on the unsigned multiplier. Currently, most research studied on the unsigned approximate multiplier, because the signed integer multipliers can be extended from the unsigned approximation methodologies. In this section, the extension of the proposed approach is to be discussed. It reveals the feasibility of the proposed approximation techniques on the signed multiplier.

2.5.1 Optimizing the Proposed Inexact Compressors for Signed Multiplier

This section focuses on the design of approximate signed multipliers with inexact compressors. To compress partial products by using low-cost circuits, an inexact m:3 compressor design is optimized for signed partial product matrix based on the proposed m:2 Com (Section 2.2), which comprises only AND and OR gates. A general expression for inexact compressors is introduced, followed by an example of a 4:3 compressor. The height of partial product matrix is reduced by compressors to three rows, which leads to fewer accumulation stages in the overall multiplier. To achieve different levels of hardware performance, three approximate signed multipliers that have almost the same accuracy are introduced.

1) Design of sign-focused m:3 compressors

								α_7	α_6	α_5	α_4	α_3	α_2	α_1	α_0
							×	β_7	β_6	β_5	β_4	β_3	β_2	β_1	β_0
							1	$\overline{\alpha_7\beta_0}$	$\alpha_6\beta_0$	$\alpha_5\beta_0$	$\alpha_4\beta_0$	$\alpha_3\beta_0$	$\alpha_2\beta_0$	$\alpha_1\beta_0$	$\alpha_0\beta_0$
							$\overline{\alpha_7\beta_1}$	$\alpha_6\beta_1$	$\alpha_5\beta_1$	$\alpha_4\beta_1$	$\alpha_3\beta_1$	$\alpha_2\beta_1$	$\alpha_1\beta_1$	$\alpha_0\beta_1$	
						$\overline{\alpha_7\beta_2}$	$\alpha_6\beta_2$	$\alpha_5\beta_2$	$\alpha_4\beta_2$	$\alpha_3\beta_2$	$\alpha_2\beta_2$	$\alpha_1\beta_2$	$\alpha_0\beta_2$		
					$\overline{\alpha_7\beta_3}$	$\alpha_6\beta_3$	$\alpha_5\beta_3$	$\alpha_4\beta_3$	$\alpha_3\beta_3$	$\alpha_2\beta_3$	$\alpha_1\beta_3$	$\alpha_0\beta_3$			
				$\overline{\alpha_7\beta_4}$	$\alpha_6\beta_4$	$\alpha_5\beta_4$	$\alpha_4\beta_4$	$\alpha_3\beta_4$	$\alpha_2\beta_4$	$\alpha_1\beta_4$	$\alpha_0\beta_4$				
			$\overline{\alpha_7\beta_5}$	$\alpha_6\beta_5$	$\alpha_5\beta_5$	$\alpha_4\beta_5$	$\alpha_3\beta_5$	$\alpha_2\beta_5$	$\alpha_1\beta_5$	$\alpha_0\beta_5$					
		$\overline{\alpha_7\beta_6}$	$\alpha_6\beta_6$	$\alpha_5\beta_6$	$\alpha_4\beta_6$	$\alpha_3\beta_6$	$\alpha_2\beta_6$	$\alpha_1\beta_6$	$\alpha_0\beta_6$						
1	$\alpha_7\beta_7$	$\overline{\alpha_6\beta_7}$	$\overline{\alpha_5\beta_7}$	$\overline{\alpha_4\beta_7}$	$\overline{\alpha_3\beta_7}$	$\overline{\alpha_2\beta_7}$	$\overline{\alpha_1\beta_7}$	$\overline{\alpha_0\beta_7}$							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 2-17 Partial product matrix of an 8-bit signed multiplier.

Consider the signed multiplication with two n-bit inputs, i.e., a multiplicand A and a multiplier B, both are in two's complement. The inputs are given as

$$A = -\alpha_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} \alpha_i \times 2^i, \tag{2-28}$$

$$B = -\beta_{n-1} \times 2^{n-1} + \sum_{j=0}^{n-2} \beta_j \times 2^j. \tag{2-29}$$

The partial product matrix for an 8-bit signed multiplier is generated using the Baugh-Wooley algorithm [60], as shown in Figure 2-17. Some partial products are generated by NAND gates, which are associated with the sign bit in operand input. Assume that the inputs A and B are uniformly and independently distributed, hence the probabilities of partial products (i.e. q) generated by AND and NAND gates can be calculated as:

$$\mathrm{P}(q_{AND} = 1) = 1/4, \tag{2-30}$$

$$\mathrm{P}(q_{NAND} = 1) = 3/4. \tag{2-31}$$

Similar with the Section 2.2.3, the partial products belonging to the same column from top to bottom are named as $q_1, q_2, \dots, q_m$,where m is the number of partial products in this column. In the signed multiplier, the arithmetic sum result of partial products in this column, R_s^m, is calculated as

$$R_s^m = \sum_1^m (q_1, q_2, \dots, q_m). \tag{2-32}$$

Table 2-12 Occurrence probability of arithmetic sum results in the signed multiplier

Probability	m **= 4**	m **= 5**	m **= 6**	m **= 7**	m **= 8**
P (R_s^m= 0)	3.52%	2.64%	1.98%	1.48%	1.11%
P (R_s^m= 1)	23.43%	18.45%	14.50%	11.37%	8.90%
P (R_s^m= 2)	46.09%	40.43%	34.94%	29.84%	25.21%
P (R_s^m= 3)	23.44%	29.10%	31.93%	32.68%	31.97%
P (R_s^m= 4)	3.52%	8.50%	13.65%	18.22%	21.84%
P (R_s^m= 5)	-	0.88%	2.78%	5.50%	8.68%
P (R_s^m= 6)	-	-	0.22%	0.86%	2.02%
P (R_s^m= 7)	-	-	-	0.05%	0.26%
P (R_s^m= 8)	-	-	-	-	0.01%

Table 2-12 shows the occurrence probability of the arithmetic sum result R_s^m when m ranges from 4 to 8., for the signed multiplier. As it can be observed in Table 2-12, the probability is low that R_s^m is greater than 3. For example, when m is 5, the probability that R_s^5 is greater than 3 is 9.38%. It shows the possibility that m partial products can be inexactly calculated using three bits. Therefore, the design of m:3 compressor (m:3 Com) is introduced to compress m partial products into three bits with the same weights.

Three output bits of m:3 Com are denoted as w_1^m, w_2^m, and w_3^m. These three bits have the same weights with the input values. Generally, the compressor is a circuit logic to count the number of '1' in the inputs. Therefore, w_1^m, w_2^m, and w_3^m are designed in turn for the cases when there is at least one, two and three '1' is in one column, respectively.

Function of the first output bit w_1^m: The first output bit is designed for the case that the number of '1' in inputs is 1. Therefore, when any one partial product is '1' among m partial products, w_1^m is '1'. The value of w_1^m can be calculated as the result of OR operation of m partial products, expressed as

$$w_1^m = q_1 + q_2 + \cdots + q_m \,. \tag{2-33}$$

Function of the second output bit w_2^m: The second output bit w_2^m is designed to calculate the number of '1' as 2 in one column. Any two partial products among m partial products are '1' meaning the number of '1' is 2. It can be calculated as the result

of AND operation of any two partial products, as

$$w_2^m = q_1 \cdot (q_2 + \cdots + q_m) + \cdots + q_i \cdot (q_{i+1} + \cdots + q_m) + \cdots + q_{m-1} \cdot q_m. \quad (2\text{-}34)$$

where, '·' means AND operation and '+' indicates OR operation.

Function of the third output bit w_3^m: The third bit is further designed on w_1^m and w_2^m to calculate the number of '1' as 3. Similar to the above analysis, w_3^m can be obtained when any three partial products are '1'. Moreover, w_3^m is specially designed for the signed multipliers. When use m:3 Com on partial product matrix shown in Figure 2-17, q_1 and q_m of m:3 Com are the partial products generated by NAND gates. The probability is 3/4 that the partial product generated by NAND gate is '1'. Therefore, we consider these two partial products specially and count the number of '1' as 3. That is (i) when q_1 and q_m both are '1', one partial product from q_2 to q_{m-1} is '1'; and (ii) when q_1 or q_m is '1', two partial products from q_2 to q_{m-1} are '1'. For the (ii), we use the AND gate on two adjacent elements to further inexactly express the condition that two partial products from q_2 to q_{m-1} are '1'. The function of w_3^m is given as:

$$w_3^m = q_1 \cdot q_{\mathrm{m}} \cdot (q_2 + \cdots + q_{m-1}) + q_1 \cdot (q_2 \cdot q_3 + \cdots + q_{\mathrm{m-2}} \cdot q_{m-1}) + q_{\mathrm{m}} \cdot (q_2 \cdot q_3 + \cdots + q_{\mathrm{m-2}} \cdot q_{m-1}). \quad (2\text{-}35)$$

2) Example of optimized 4:3 compressor

In this subsection, an example of the sign-focused m:3 Com is shown to illustrate the optimized compressor in detail.

With the help of Eq. 2-29, 2-30 and 2-31, the function of 4:3 Com can be expressed as:

$$w_1^4 = q_1 + q_2 + q_3 + q_4,$$
$$w_2^4 = q_1 \cdot (q_2 + q_3 + q_4) + q_2 \cdot (q_3 + q_4) + q_3 \cdot q_4,$$
$$w_3^4 = q_1 \cdot q_4 \cdot (q_2 + q_3) + q_1 \cdot (q_2 \cdot q_3) + q_4 \cdot (q_2 \cdot q_3). \quad (2\text{-}36)$$

Figure 2-18 shows the structure of 4:3 Com based on Eq. 2-32. An XOR gate costs more power and delay than AND and OR gates. Although the function of 4:3 Com seems complex, it only consists of AND and OR gates. These basic gates are easily to be synthesized to power- and area-efficient compound logic cell.

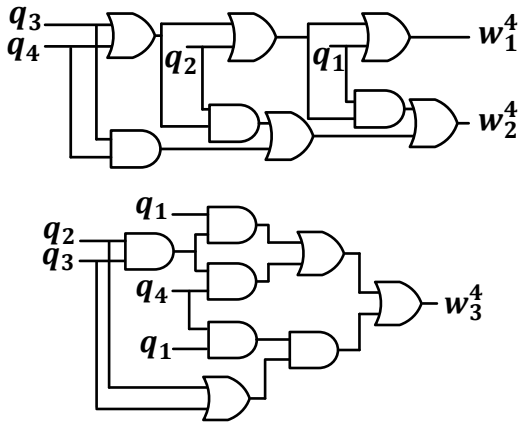


Figure 2-18 The structure of optimized 4:3 Com.

Table 2-13 The behavior of 4:3 Com

q_1	q_2	q_3	q_4	w_1^4	w_2^4	w_3^4	R_s^4	$\widetilde{R_s^4}$	Probability
0	0	0	0	0	0	0	0	0	0.03516
0	0	0	1	1	0	0	1	1	0.10547
0	0	1	0	1	0	0	1	1	0.01172
0	0	1	1	1	1	0	2	2	0.03516
0	1	0	0	1	0	0	1	1	0.01172
0	1	0	1	1	1	0	2	2	0.03516
0	1	1	0	1	1	0	2	2	0.00391
0	1	1	1	1	1	1	3	3	0.01172
1	0	0	0	1	0	0	1	1	0.10547
1	0	0	1	1	1	0	2	2	0.31641
1	0	1	0	1	1	0	2	2	0.03516
1	0	1	1	1	1	1	3	3	0.10547
1	1	0	0	1	1	0	2	2	0.03516
1	1	0	1	1	1	1	3	3	0.10547
1	1	1	0	1	1	1	3	3	0.01172
1	1	1	1	1	1	1	4	3×	0.03516

* R_s^4 indicates the exact sum result of four partial products, calculated as $R_s^4 = \sum(p_1, p_2, p_3, p_4)$.

*w_1^4, w_2^4, and w_3^4 are the output bits of 4:3 Com. $\widetilde{R_s^4}$ means the inexact sum result.

Table 2-13 shows the behavior of 4:3 Com, where $\widetilde{R_s^4}$ indicates the inexact sum result, given by

$$\widetilde{R_s^4} = \sum(w_1^4, w_2^4, w_3^4). \tag{2-37}$$

It can be seen in Table 2-12, the 4:3 Com can produce the exact sum result for most input combinations. The error occurs when the input combination is '1111', and the probability is 3.52%.

2.5.2 Extension to Signed Approximate Multiplier

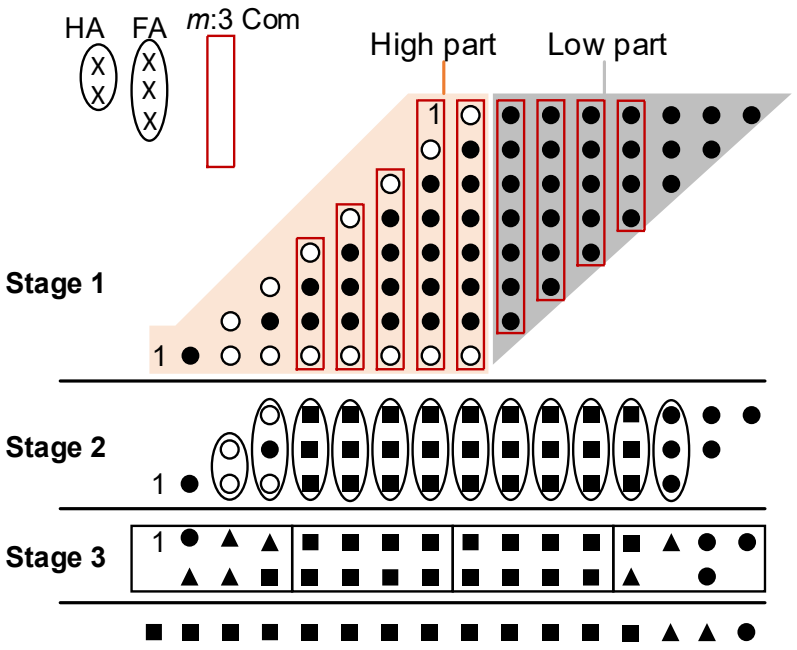
In this section, similarly, m:3 Coms are selectively applied on the partial product matrix to accumulate partial products in the signed multiplier. This step corresponds to common technique of the inexact partial product accumulation. Then, a carry-save adder (CSA) is used to accumulate results from compressors to fed to the final addition. In the last step, a carry look-ahead adder (CLA) is employed to produce the final multiplication result.

Figure 2-19 shows the structure of approximate signed multiplier design. The partial product matrix is divided into two parts according to the bit significances. High part includes the partial products from bit 7 to 15, while low part involves the partial products from bit 0 to 6.

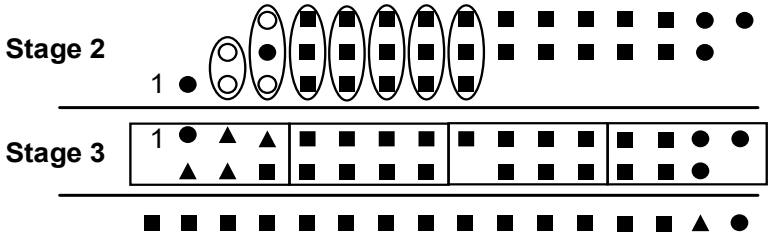
Because the m:3 Com is specially designed for partial products associated with the sign bit (these partial products are only on high part), the functionality of m:3 Com is fully utilized on high part. For the operation of low part, three types of approximate multipliers with sign-focused compressors (AMSC) are proposed:

a) AMSC1

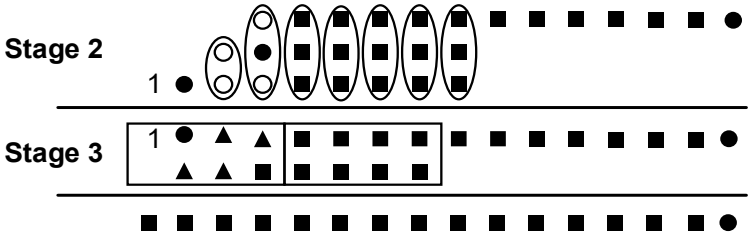
As shown in Figure 2-19 (a) of AMSC1, in the Stage 1, the functionality of m:3 Com is completely used on partial product matrix. Eight rows of partial products are reduced into three rows by m:3 Coms. In Stage 2, a CSA includes ten full-adders (FAs) and one half-adder (HA). Finally, a CLA gives the final multiplication result. The



(a)



(b)



(c)

Figure 2-19 The structure of 8-bit approximate signed multipliers. Stage 1 is partial product accumulation step, where ● and ○ indicates the partial products generated by AND and NAND gates, respectively. Exact HA and FA are used as CSA step in Stage 2, where ▲ and ■ means the exact and inexact elements, respectively. A CLA is employed in Stage 3. (a) AMSC1: The functionality of m:3 Com is fully used in Stage 1. (b) AMSC2: Only the first and second output bits of m:3 Com are used on low part in Stage 1. (c) AMSC3: Only the first output bit is used on low part in Stage 1.

accumulation stages and circuit complexity are reduced significantly by the proposed compressors that comprise only AND and OR gates.

b) AMSC2

The third output bit of m:3 Com is specially designed for the partial products generated by NAND gates. Partial products on low part are all generated by AND gates, hence the third output bit is not necessary for low part. As shown in Figure 2-19 (b), the first and the second output bits are used to represent the result of accumulation on low part, while the third output bit is ignored. Note that, AMSC2 needs one 3:3 Com on bit 2. Therefore, eight rows of partial products are accumulated into three rows on high part and two rows on low part. The adders in CSA stage are reduced with the decreased elements in Stage 2.

c) AMSC3

Compared with AMSC1 and AMSC2, the approximation is further applied on AMSC3. As shown in Figure 2-19 (c), only the first output bit of m:3 Com is remained to represent the result of low part. In addition, one OR gate is used on bit 1. Therefore, eight rows of partial products are reduced into three rows on high part and one row on low part, by fully using m:3 Com and partly using m:3 Com, respectively.

2.5.3 Experiment for Signed Approximate Multipliers

In this section, the accuracy and hardware performance of the signed multipliers are evaluated. To investigate the trade-off between accuracy loss and hardware gaining, the proposed multipliers were implemented and compared with the exact signed multiplier and approximate signed multiplier introduced in [37] (R4ABM1-k and R4ABM2-k). Note that the parameter k of the designs in [37] is the bit-size of approximation from the lowest significant bit.

a) Accuracy analysis

The error distance (ED), the normalized worst case error (NWCE) and the error rate (ER) are defined in Eq. 2-14 (Section 2.4.2). The normalized error distance (NED) are defined to evaluate the averaging effect of a set of output for a multiplier [57]. The NED is useful in the reliability assessment of a design and is defined as MED normalized by the maximum possible error.

Table 2-14 Accuracy comparisons for signed approximate multipliers

Designs	NED (%)	NWCE (%)	ER (%)
AMSC1	0.99	12.12	77.29
AMSC2	1.01	12.31	77.51
AMSC3	1.10	12.50	87.22
R4ABM1-10 [37]	0.70	4.29	78.15
R4ABM1-12 [37]	2.07	13.00	78.69
R4ABM1-14 [37]	5.20	33.02	78.81
R4ABM2-10 [37]	0.78	5.34	96.12
R4ABM2-12 [37]	2.46	16.28	96.57
R4ABM2-14 [37]	6.33	41.28	96.66

Table 2-14 shows the accuracy comparisons for different signed multipliers in terms of NED, NWCE and ER. Because the different functionality of the proposed compressor is used, the proposed multipliers can achieve different levels of accuracy. In addition, the accuracies of three proposed multipliers are close. In terms of ER, AMSC1 has the lowest value among all multipliers as 77.29%. R4ABM1-10 has the lowest accuracy losses in terms of NED and NWCE.

b) Hardware performance analysis

Exact signed multiplier [61], previous approximate multipliers [37], and the proposed design in this work were implemented in Verilog HDL and synthesized by using the Synopsys Design Complier. All designs were evaluated at the same condition with Section 2.4.1.

Table 2-15 Hardware performance of signed multipliers

Designs	Power (uW)	Area (um^2)	Delay (ns)	PDP (fJ)
Exact	235.9	531.47	2.71	639.29
AMSC1	112.2	329.73	2.11	236.74
AMSC2	91.8	283.13	1.72	157.90
AMSC3	66.3	205.88	1.52	100.78
R4ABM1-10 [37]	165.7	398.68	2.25	372.83
R4ABM1-12 [37]	150.0	370.27	2.07	310.50
R4ABM1-14 [37]	142.1	355.91	2.07	294.15
R4ABM2-10 [37]	166.5	376.34	1.86	309.69
R4ABM2-12 [37]	151.4	341.54	1.82	275.55
R4ABM2-14 [37]	141.9	317.28	1.72	244.07

Table 2-15 shows the power, area, delay and power-delay product (PDP) for all signed multipliers. As it can be observed, the proposed approximate multiplier design gives the significant advantages compared with the exact signed multiplier. Particularly, AMSC3 achieves the lowest hardware consumption. In terms of area improvement, the proposed design can reduce area of the exact multiplier by 37.96%~61.26%. Compared with the exact one, the proposed design achieves 22.14%~43.91% delay saving. The proposed most accurate multiplier (AMSC1) can save power of the exact multiplier by 52.44%.

For these improvements by the proposed multipliers, we can consider the construction of multiplier to explain. In the experiment, we chose the modified Booth multiplier as the exact one, which needs one partial product generation stage with Booth encoding, one exact 4:2 compressor stage, one CSA stage and one CLA. In [37], there are one inexact partial product generation stage, one exact 4:2 compressor stage and one CLA. The 4:2 compressors are serial both in exact multiplier and in multipliers of [37]. These two designs employ XOR gates in each stage. On the contrast, the proposed design includes partial product generation stage using AND and NAND gates, one inexact m:3 Com stage, one CSA stage and one CLA. The m:3 Coms process in parallel, and they comprise only AND and OR gates, which can be synthesized into compound gates. For example, Figure 2-20 shows the schematic for 4:3 Com obtained after synthesis. Compared with the original structure of 4:3 Com shown in Figure 2-18,

the structure is simpler. Some AND and OR gates are synthesized into compound gate, such as OAI21. It demonstrates the validity of the m:3 Com whose feature is no-XOR gate.

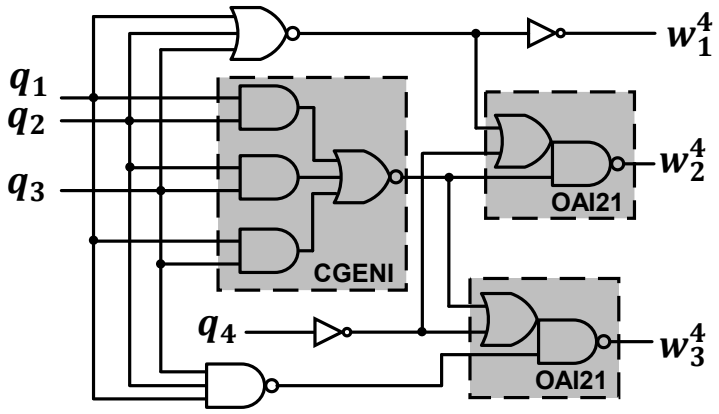


Figure 2-20 Schematic for 4:3 Com. The dotted block with gray background indicates the compound gate cell.

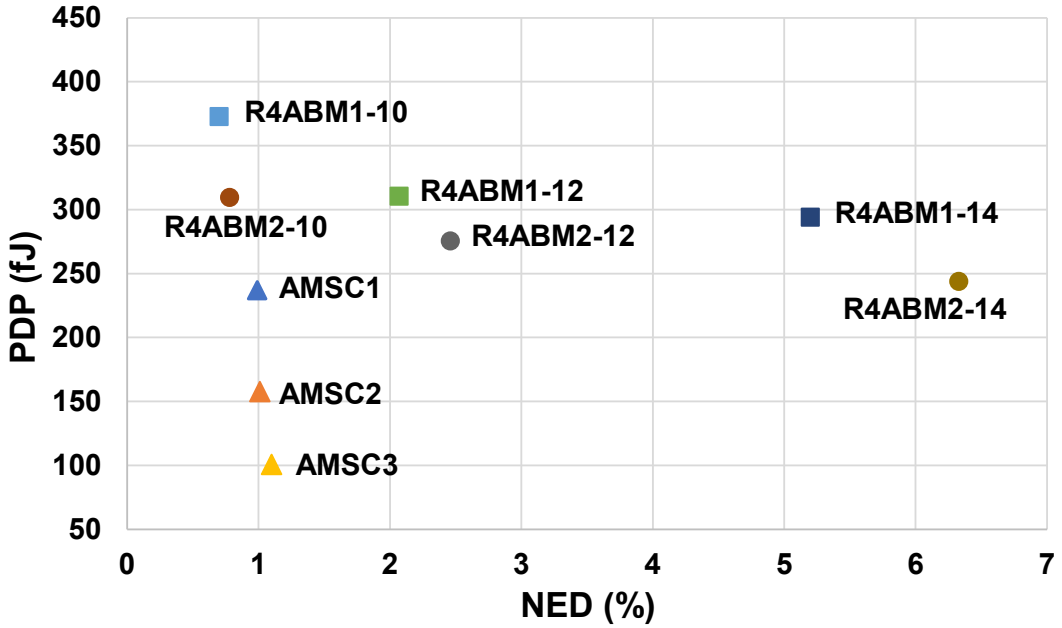


Figure 2-21 PDP versus NED for approximate signed multipliers.

An intuitive comparison for all approximate multipliers in terms of PDP and NED is shown in Figure 2-21. R4ABM1, R4ABM2 and the proposed multipliers all have various NED-PDP configurations. The accuracy ranges of R4ABM1 and R4ABM2

both are larger than that of the proposed design. However, the energy is becoming the first concern when the accuracy loss is acceptable. The PDPs of the proposed multiplier design change drastically with the configurations changing. AMSC1, AMSC2 and AMSC3 all have NED around 1%, while they deliver the PDP range from 100.72 fJ to 236.74 fJ. Moreover, when the NED is around 1%, the proposed multipliers provide the lower PDP than R4ABM1-10 and R4ABM2-10.

2.6 Summary

In this chapter, an approximate multiplier design is proposed with inexact compressors that costs lower hardware consumption than accurate multiplier. Firstly, inexact compressors with no-XOR gates for partial product reduction step is proposed. Then, an 8×8 multiplier is divided into three blocks. To employ high precision operations on significant bits and use low precision operations on insignificant bits, three different precision operations are applied. Finally, to improve accuracy, a grouped error recovery scheme with a shorter critical path is introduced. The theoretical analysis on area and delay is provided for the proposed multipliers. Moreover, the experimental results demonstrate that the proposed multiplier design significantly reduces the hardware consumption of exact multiplier. The power reduction is 59.75%~70.75%, the delay reduction is 12.78%~52.42%, and the area reduction is 42.47%~50.97%. Compared with the state-of-the-art ASIC-based approximate multipliers, the proposed design delivers more hardware reduction under a comparable accuracy. The proposed inexact compressors are optimized for signed approximate multiplier, which shows the feasibility of the proposed approximation.

3. FPGA-Based Approximate Multiplier using Carry-Inexact Elementary Modules

In this chapter, low-cost FPGA-based approximate multiplier design, is to be discussed, which is designed to achieve the energy efficiency by carefully considering the construct of FPGA.

The background and necessity of FPGA-based approximate multipliers are presented in Section 3.1. The motivations driven from the existing issues and contributions of this work are also stated in this section.

The preliminaries of FPGA are to be introduced in Section 3.2 for easily understanding the FPGA-fabric utilized in this research.

Section 3.3 presents the three types of approximate 4×4 multipliers with different performances. The occurrence probability of carry from current bit to higher bit is analyzed first. The calculation of carry result in these three multipliers are approximated to shorten the critical path and reduce the circuit complexity.

By taking the proposed approximate 4×4 multipliers as the elementary modules, the large size multiplier can be constructed from small size multipliers. Section 3.4 introduces the design of approximate 8×8 multipliers. To fast produce the final product, two inexact additions are proposed.

Section 3.5 discusses the evaluation of 4×4 multipliers and 8×8 multipliers. Firstly, the accuracy and hardware performance of 4×4 multipliers are to be evaluated. Then, all configurations of the proposed 8×8 multipliers are discussed, and eight configurations are selected to further evaluate and compare with existing approximate multipliers. Moreover, to demonstrate the efficiency of the proposed approximate multiplier design, Pareto-optimal analysis is to be conducted in this section. In terms of mean relative error distance (MRED), the error of the proposed 8×8 multiplier is as low as 1.06%. Compared with the exact multiplier, the proposed design can reduce area by 43.66% and power by 24.24%. The critical path latency reduction is up to 29.50%.

The proposed multiplier design has a better accuracy-hardware result than other designs with comparable accuracy. Finally, image sharpening processing is used to assess the efficiency of approximate multipliers on application.

Section 3.6 concludes the FPGA-based approximate multipliers.

3.1 Introduction

3.1.1 Background

There are increasing demands of power- and area-efficient designs for lots of applications such as multimedia processing, data mining and machine learning. For most of these applications, approximate multiplier design has been considered as a potential approach to reduce energy by exploiting the exactness relaxation in error-tolerant applications.

In order to provide hardware-efficient and high-performance multipliers, previous works have proposed various designs of approximate multipliers, mainly for ASIC-based systems. Here are the review of state-of-the-art approximate ASIC-based multipliers related to this part. An approximation technique employing Karnaugh map for the multiplier has been discussed in [55]. Based on [55], several variants of approximate addition and multiplication units have been discussed in [63]. An open-source library of 8×8 approximate adders and multipliers, EvoApprox8b, has been presented in [64], by utilizing different approximate adders and multipliers from literatures.

However, due to the non-reconfiguration and slow development round, ASIC-based approximate multipliers are usually dedicated for one particular application and not efficient for extensive applications. In contrast, FPGA has been a promising platform for lots of applications, because it has advantages of high energy efficiency, capability of reconfiguration and fast development round [65].

Unfortunately, little study has been conducted to FPGA-specific approximate multiplier design [46][47][66][67]. Two state-of-art FPGA-based approximate

multipliers are proposed in [46][47], while exact adder used in them might cause high power and latency consumptions. In [46], three designs for approximate $n \times n$ multipliers are proposed, and large multipliers are constructed from four $n \times n$ multipliers along with exact adder. In [47], an inexact 4×2 multiplier is proposed by using four LUTs; then, approximate 4×4 multipliers and 8×8 multipliers (Cc, Ca) are constructed from 4×2 multipliers. In Ca, the adder for summing small-size multipliers is exact. The exact adder in [46][47] has serial carry propagation path, which still causes high latency and energy consumptions.

3.1.2 Necessity of FPGA-Based Approximate Multipliers

In this section, the necessity of FPGA-based approximate design is stated as follows. Also, the necessity of research on LUT-based operation is presented.

1) Necessity of FPGA-based approximate design

FPGA has been a promising platform for lots of applications, because it has advantages of high energy efficiency, capability of reconfiguration and fast development round. It is common to employ FPGA as accelerator for many applications which have lots of multiplications. Therefore, it is expected to design low-cost FPGA-specific approximate multipliers.

However, due to the architectural differences between ASICs and FPGAs, the savings achieved by ASIC-based defined designs might not comparably translate to the savings on FPGA-based implementation. In another word, the ASIC-based approximation techniques are less effective like energy, latency and area, when used for FPGA-based system.

A comparison of ASIC-based and FPGA-based implementation for ASIC-based defined approximate multiplier has been reported in [48], which pointed out this issue. Figure 3-1 illustrates this comparison results for four multipliers, in which D1-D4 were randomly selected from EvoApprox8b library [64] and a variant of approximate multiplier was selected as D5 from [63]. The gains in y-axis indicates the performance

gains of different approximate multipliers compared with an exact multiplier implementation. It can be observed that, the performance gains reported for ASIC-based implementations, are not comparably translated for FPGA-based implementations. The primary reason for this deviation is the architectural differences between ASICs and FPGAs. Logic gates are main factors should be considered when designing ASIC-based approximate multiplier. However, FPGAs consists of completely different entities, that is, look-up-tables (LUTs) and carry chains. Consequently, any approximation techniques, optimized for FPGA-based systems, must consider the structure of FPGA-fabric.

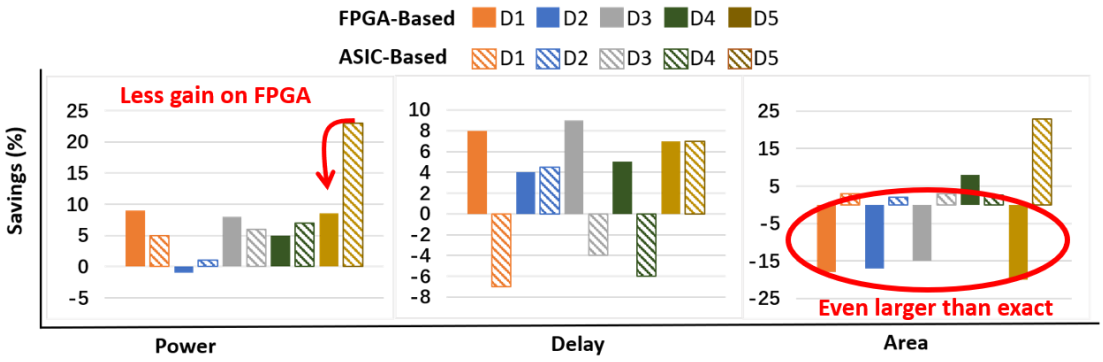


Figure 3-1 A comparison of ASIC-based implementation and FPGA-based implementation for five state-of-the-art ASIC-based approximate multipliers [48].

2) Necessity of research on LUT-based operation

Furthermore, on modern FPGA, both DSP blocks and LUT-based blocks are necessary when performing arithmetic operations. The statistics reported in [47] have shown the necessity of LUT-based operations, the results are illustrated in Table 3-1. The authors compared two applications of Reed-Solomon and JPEG encoders with LUTs and DSP blocks using Xilinx Vivado 17.1 for Virtex-7 series FPGA (7VX330T device).

Here are two observations can be concluded: (i) The delay for DSP blocks enabled situation is larger than that of disabled situation, because the location of the allocated DSP blocks incurs the routing delay. For small-size applications, to improve the overall

performance of an application, manual Floorplanning is feasible to be processed. However, for complex or large-size applications that have competitive requirements for FPGA resources, it is difficult to process the placement optimization on the required FPGA resources to improve performance. (ii) As shown in Table 3-1, the implementation of the JPEG-encoder has a large number of DSP blocks, which corresponds to 56% of the total available DSP blocks. Such applications could exhaust the available DSP blocks for critical operations, while other applications on the same FPGA will trend to use the LUT-based blocks [68].

Table 3-1 Comparison of DSP blocks and LUTs based implemenations [47].

Design	DSP blocks enables			DSP blocks disabled		
	Delay [ns]	#LUTs	#DSP blocks	Delay [ns]	#LUTs	#DSP blocks
Reed-Solomon encoder	5.115	2826	22	4.358	2867	0
JPEG encoder	8.637	71362	631	9.732	14780	0

This is why, despite the availability of DSP blocks, Xilinx and Intel also provide LUT-based multipliers [69][70]. Moreover, 8-bit integer multiplier is synthesized as LUT-based implementation under default conditions. Therefore, operations with LUT-based blocks are valuable to be explored.

3.1.3 Research Motivations and Contributions

Motivated by the demand of approximate multiplier for FPGA-based systems, as pointed out in Section 3.1.2, a novel methodology for designing approximate multipliers by employing the FPGA-based fabrics (primarily look-up tables and carry chains), is discussed in this chapter. In addition, to solve the problem of slight energy and latency savings in existing FPGA-based approximate multipliers, as pointed as in Section 3.1.1, this work aims to propose an FPGA-based multiplier design with low-cost. Moreover, considering the configurability of FPGA, wide-range of multiplier configurations are also discussed in this work.

In this work, low-cost FPGA-based approximate multipliers are proposed, whereas most of previous works focused on ASIC-based approximate multipliers. Different from the exact adder in FPGA-based multipliers in [41][42], this proposed design introduces two inexact adders for lowering hardware cost on FPGA. The carry propagation path usually occupies the primary hardware consumptions in the multiplier, hence this work focuses on the approximation of carry results. The primary contributions are as follows:

i) **Three types of approximate 4×4 multipliers** implemented with LUTs and associated carry chain, are proposed in this work. The critical path is shortened by restricting the carry generation in the multiplier.

ii) **The large-size multipliers** are exploited on architectural space and **provide a wide-range of approximate 8×8 multipliers** by using the proposed 4×4 multipliers as elementary modules. Eight configurations for approximate 8×8 multipliers are presented for different accuracy-hardware requirements.

Figure 3-2 illustrates the overview of methodologies in this work. Necessary preliminaries of this work are introduced in Section 3.2. In this work, the LUTs and carry chain are employed for the proposed approximate multiplier, hence these two structures are introduced detailly. Three types of approximate 4×4 multipliers are proposed in Section 3.3, where the carry of accumulation is restricted. Then, in Section 3.4, take the proposed 4×4 multipliers as elementary modules, by utilizing the architectural-space construction, large approximate multipliers are constructed from the proposed 4×4 multipliers. In addition, to reduce the delay consumption of the multiplier, two inexact additions are also proposed in Section 3.4, which are used to sum the products from small multipliers.

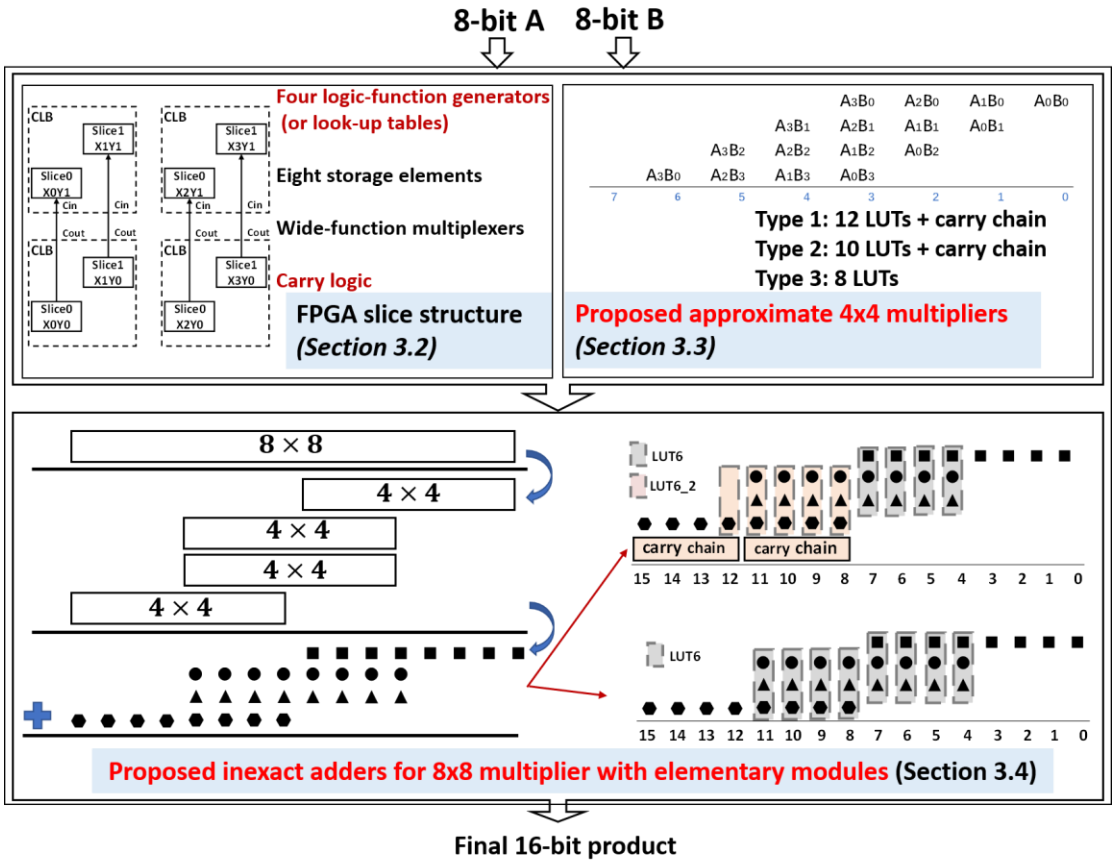


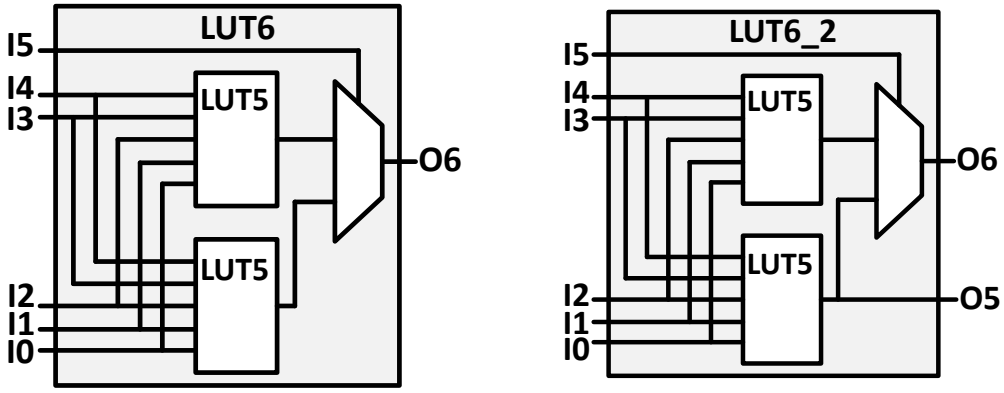
Figure 3-2 Overview of the FPGA-based approximate multiplier.

3.2 Preliminaries of FPGA-fabric

This works targets the devices of Xilinx 7-series FPGA family. The proposed design can also be implemented on FPGAs from other vendors, which provide 6-input LUTs and carry chains.

The configurable logic blocks (CLBs) are the main logic resources for implementations of sequential as well as combinational circuits, and one CLB consists of two slices. Each slice has four 6-input look-up tables (LUTs), eight storage elements to register the outputs of LUTs, wide-function multiplexers, and a fast 4-bit carry chain [64]. A 6-input LUT can be configurated as one of the following two implementations. One implementation is a single 6-input combinational function with one output O6 as shown in Figure 3-3 (a), commonly referred as LUT6. Another implementation is

named as LUT6_2, which has two 5-input combinational functions with O5 and O6 outputs as shown in Figure 3-3 (b).



(a) (b)

Figure 3-3 The structure of 6-input LUT [71].

A LUT is instantiated with an INIT attribute which specifies the logic function of one LUT. An INIT attribute consists of 16 hexadecimal values (i.e. 64 binary values for 64 input combinations). The INIT value can be determined by creating a binary logic table of all input combinations. It indicates that the logic value '1' occurs on the outputs among all 64 combinations. For example, as shown in Figure 3-4, only the output for input combination '000010' is '1'. From the bottom combination to the top combination, the value of output is '0000000000000004' (hex). This hexadecimal value is the INIT value for the function of LUT6, it means that the output O6 is '1' for the input combination '000010'.

I5	I4	I3	I2	I1	I0	O6
0	0	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	0	1	0	1
0	0	0	0	1	1	0
.	.				.	.
.	.				.	.
.	.				.	.
.	.				.	.
1	1	1	1	1	0	0
1	1	1	1	1	1	0

The value from bottom to top is '00……0100' (bin)

'0000000000000004' (hex)

Figure 3-4 Example of INIT value for LUT6.

The structure of the carry chain is shown in Figure 3-5. The outputs of LUT drive the inputs of the carry chain. It comprises multiplexers with bypass signals (AX/BX/CX/DX) and XOR gates. The carry chain usually implements as a 4-bit carry-look ahead adder to perform fast function with O5 as carry-generate signal and O6 as carry-propagate signal.

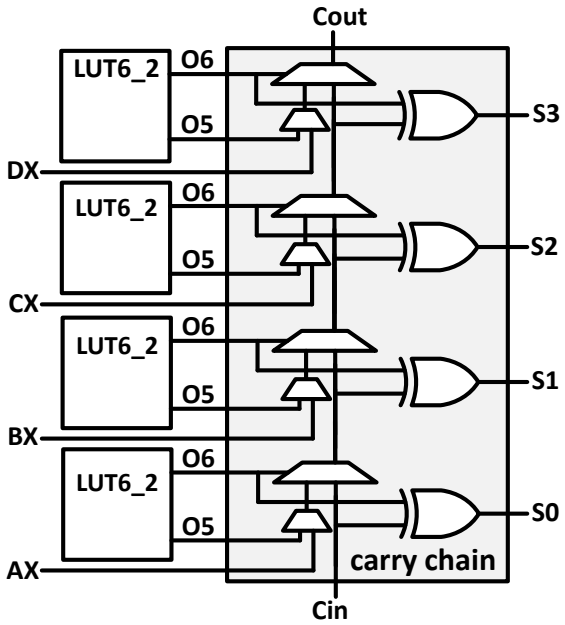


Figure 3-5 The structure of carry chain [71].

3.3 Proposed Approximate 4×4 Multipliers

This work presents three novel approximate 4×4 multiplier designs which provide different accuracy-hardware tradeoffs. The occurrence probability of carry is analyzed in Section 3.3.1. Based on low probability of carry, three approximate multipliers are introduced with no-carry compressors. The first multiplier with low-error feature is introduced in Section 3.3.2. The second multiplier has an optimized structure on the first design and is introduced in Section 3.3.3. Section 3.3.4 presents the third design which is implemented by only LUTs.

3.3.1 Occurrence Probability of Carry

In the FPGA-based multiplier, the compressors in partial product accumulation are implemented by LUTs, and their inputs are partial product. Each partial product is generated by AND gate. Assume that two inputs of the multiplier are uniformly and independently distributed, hence the probability of one partial product equaling to '1' is 0.25. Figure 3-6 shows the occurrence probability of each column in 4×4 multiplier. It can be observed that the carry is generated in rare cases. Therefore, by applying approximation on carry could save the LUT and also guarantee the accuracy.

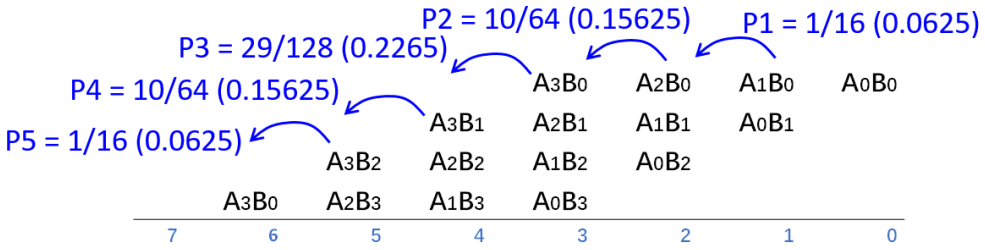


Figure 3-6 Occurrence probability of carry in 4×4 multiplier.

3.3.2 Approximate 4×4 Multiplier 1 (AFM1)

The high complexity of the multiplier is usually caused by the carry propagation path during accumulating partial products. To reduce the complexity of the multiplier, we do approximation on the carry signals.

The structure of approximate 4×4 multiplier 1 (AFM1) are shown in Figure 3-7, where three layers of LUTs are used to accumulate partial products and the carry chain is used to produce the multiplication result. The expression of each LUT in AFM1 is shown in Table 3-2.

In AFM1, Layer 1 computes the carry result from the preceding column. For example, column 4 consists of six elements B_3, B_2, B_1, A_3, A_2 and A_1. LUT_9 is fully used to compute the carry result from column 4, that is, six inputs of LUT_9 are

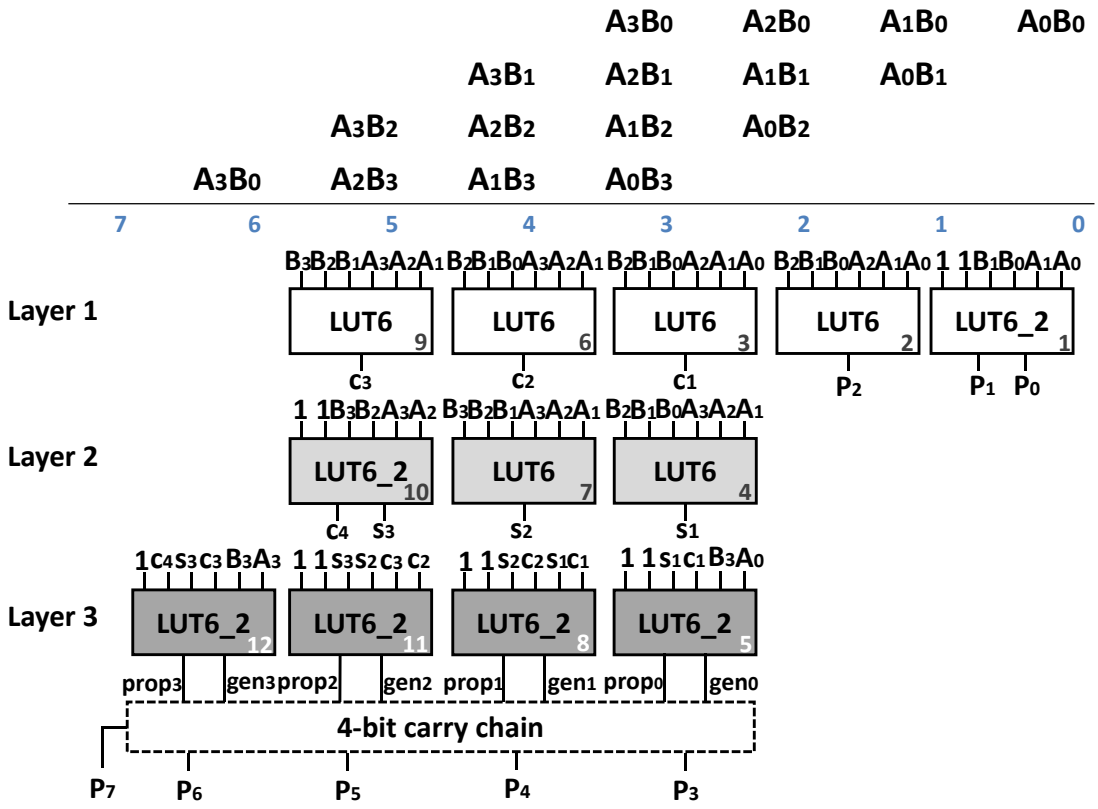


Figure 3-7 The structure of AFM1. Layer 1 computes the carry result from the preceding column while Layer 2 generates the sum result for the current column. Layer 3 produces the carry-propagate and carry-generate signals for the carry chain.

completely used for six elements. However, there are eight elements on column 3, which exceeds the input number of one 6-input LUT. Therefore, LUT_6 inexactly computes the carry result from column 3 by ignoring one partial product. The carry result from columns 1 and 2 is computed by LUT_3. When B_2, B_1, B_0, A_2, A_1 and A_0 on columns 1 and 2 all are '1', the exact carry result is 2-bit '10' (bin). In AFM1, this carry result is inexactly computed as 1-bit '1' (bin) by LUT_3.

Layer 2 computes the sum result generated from the current column, while Layer 3 produces the carry-propagate and carry-generate signals for the associated carry chain. Particularly, to fully utilize the LUT resource, a LUT6_2 with two outputs (i.e. LUT_{10}) is employed in Layer 2 to generate both the carry result (i.e. c4) and the sum result (i.e. s3) from column 5. Table 3-3 illustrates the expression of each LUTs in AFM1.

Table 3-2 Input and output configurations for each LUT in AFM1.

LUT	Input configuration						Output configuration		INIT value (Hex)
	I5	I4	I3	I2	I1	I0	O6	O5	
LUT1	1	1	B_1	B_0	A_1	A_0	P_1	P_0	6AC06AC0A0A0A0A0
LUT2	B_2	B_1	B_0	A_2	A_1	A_0	P_2		1E665AAAB4CCF000
LUT3	B_2	B_1	B_0	A_2	A_1	A_0	c_1		E888A000C8000000
LUT4	B_2	B_1	B_0	A_3	A_2	A_1	s_1		96665AAA3CCCF000
LUT5	1	1	s_1	c_1	B_3	A_0	$prop_0$	gen_0	8778877808800880
LUT6	B_2	B_1	B_0	A_3	A_2	A_1	c_2		E888A000C0000000
LUT7	B_3	B_2	B_1	A_3	A_2	A_1	s_2		96665AAA3CCCF000
LUT8	1	1	s_2	c_2	s_1	c_1	$prop_1$	gen_1	8778877808800880
LUT9	B_3	B_2	B_1	A_3	A_2	A_1	c_3		E888A000C0000000
LUT10	1	1	B_3	B_2	A_3	A_2	c_4	s_3	800080006AC06AC0
LUT11	1	1	s_3	s_2	c_3	c_2	$prop_2$	gen_2	936C936C20802080
LUT12	1	c_4	s_3	c_3	B_3	A_3	$prop_3$	gen_3	87777888F8888000

Table 3-3 The expression of LUTs in AFM1

LUT	INIT value (Hex)	Expression
LUT1	6AC06AC0A0A0A0A0	$P_0 = A_0B_0$ $P_1 = (A_1B_0) \oplus (A_0B_1)$
LUT2	1E665AAAB4CCF000	$P_2 = (A_1A_0B_1B_0) \oplus (A_2B_0) \oplus (A_1B_1) \oplus (A_0B_2)$
LUT3	E888A000C8000000	$c_1 = A_2A_1B_1B_0 + A_2A_0B_2B_0 + A_1A_0B_2B_1 + A_1A_0B_1B_0$
LUT4	96665AAA3CCCF000	$s_1 = (A_3B_0) \oplus (A_2B_1) \oplus (A_1B_2)$
LUT5	8778877808800880	$prop_0 = (s_1 \oplus c_1) \oplus (A_0B_3)$ $gen_0 = (s_1 \oplus c_1)(A_0B_3)$
LUT6	E888A000C0000000	$c_2 = A_3A_2B_1B_0 + A_3A_1B_2B_0 + A_2A_1B_2B_1$
LUT7	96665AAA3CCCF000	$s_2 = (A_3B_1) \oplus (A_2B_2) \oplus (A_1B_3)$
LUT8	8778877808800880	$prop_1 = (s_2 \oplus c_2) \oplus (s_1c_1)$ $gen_1 = (s_2 \oplus c_2)(s_1c_1)$
LUT9	E888A000C0000000	$c_3 = A_3A_2B_2B_1 + A_3A_1B_3B_1 + A_2A_1B_3B_2$
LUT10	800080006AC06AC0	$s_3 = (A_3B_2) \oplus (A_2B_3)$ $c_4 = A_3B_2A_2B_3$
LUT11	936C936C20802080	$prop_2 = (s_3 \oplus c_3) \oplus (s_2c_2)$ $gen_2 = (s_3 \oplus c_3)(s_2c_2)$
LUT12	87777888F8888000	$prop_3 = (s_3c_3) \oplus (A_3B_3) \oplus c_4$ $gen_3 = (s_3c_3) \oplus (A_3B_3)c_4 + s_3c_3A_3B_3$

Table 3-4 Error occurrences of AFM1 and Ca [47].

Design	Input combinations	Exact result	Approximate result	Difference
AFM1	7×7	49	41	8
	7×15	105	97	8
	15×7	105	97	8
	15×15	225	217	8
Ca [47]	5×15	75	67	8
	6×7	42	34	8
	6×15	90	82	8
	7×15	105	97	8
	13×13	169	161	8
	15×5	75	67	8

In AFM1, Layer 1 is inexact by approximating the carry and Layer2 is also inexact by partly calculating the sum results but the Layer 3 could recover the approximation in Layer 2. Table 3-4 illustrates the error occurrences of AFM1. Assume that two inputs of the multiplier are uniformly and independently distributed. The maximum error distance is '8' for all input combinations and the error probability of AFM1 is 0.0156 (= 4/256). The total number of LUTs in AFM1 is 12 and the critical path consists of 2 LUTs along with a carry chain.

3.3.3 Approximate 4×4 Multiplier 2 (AFM2)

Approximate 4×4 multiplier 2 (AFM2) is proposed by optimizing the structure of AFM1 to further reduce the area.

Figure 3-8 shows the structure of AFM2 which has the similar structure with AFM1. The carry result from preceding column and the sum result for the current column is computed by Layer 1 and Layer 2, respectively. For a uniform and independent distribution of the inputs of a 4×4 multiplier, the probability is as low as 7/64 that the carry is generated from columns 1 and 2. Therefore, the carry result from columns 1 and 2 is omitted by eliminating LUT_3 in AFM1. Another optimization is eliminating LUT_6 in AFM1, where both the carry result and the sum result from column 3 are computed by one LUT6_2 (i.e. LUT_3 in AFM2). The functions of LUT_5 and

LUT7~LUT10 in AFM2 are the same as those of LUTs in the same positions in AFM1.

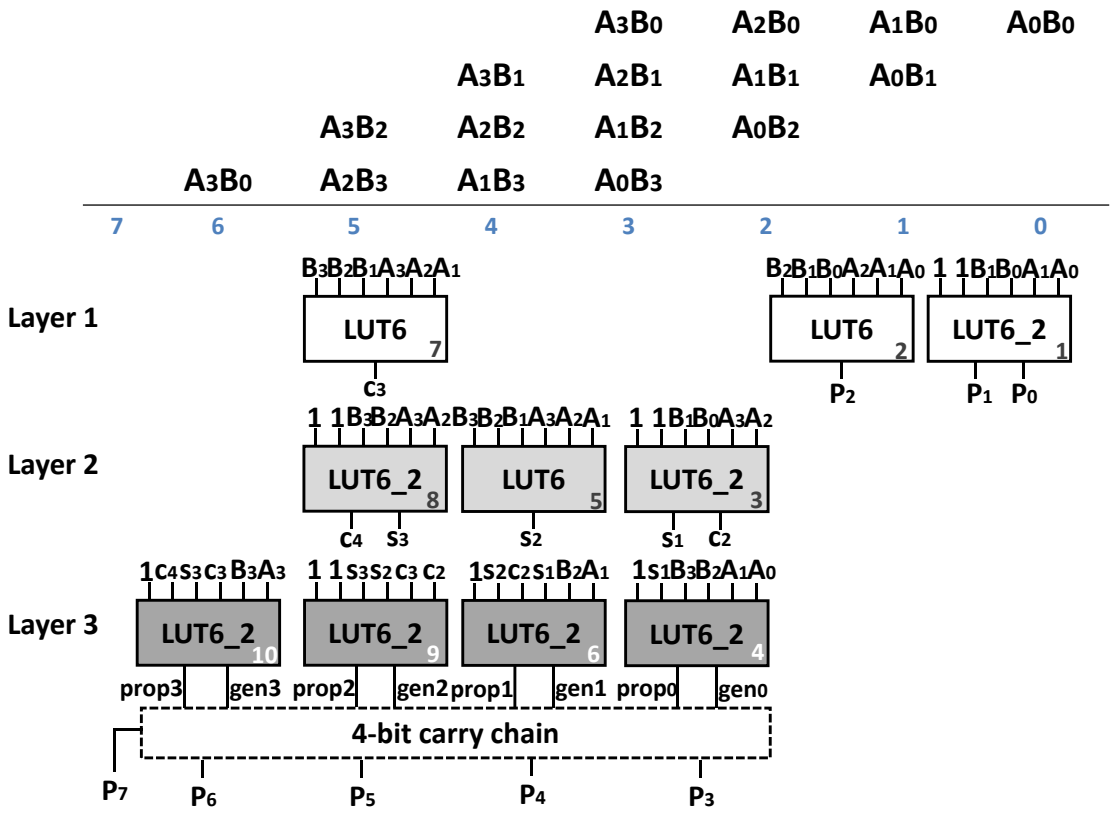


Figure 3-8 The structure of AFM2.

Table 3-5 Input and output configurations for each LUT in AFM2.

LUT	Input configuration						Output configuration		INIT value (Hex)
	I5	I4	I3	I2	I1	I0	O6	O5	
LUT1	1	1	B_1	B_0	A_1	A_0	P_1	P_0	6AC06AC0A0A0A0A0
LUT2	B_2	B_1	B_0	A_2	A_1	A_0	P_2		1E665AAAB4CCF000
LUT3	1	1	B_1	B_0	A_3	A_2	s_1	c_2	6AC06AC080008000
LUT4	1	s_1	B_3	B_2	A_1	A_0	$prop_0$	gen_0	953F6AC02A008000
LUT5	B_3	B_2	B_1	A_3	A_2	A_1	s_2		96665AAA3CCCF000
LUT6	1	s_2	c_2	s_1	B_2	A_1	$prop_1$	gen_1	807F7F8000808000
LUT7	B_3	B_2	B_1	A_3	A_2	A_1	c_3		E888A000C0000000
LUT8	1	1	B_3	B_2	A_3	A_2	c_4	s_3	800080006AC06AC0
LUT9	1	1	s_3	s_2	c_3	c_2	$prop_2$	gen_2	936C936C20802080
LUT10	1	c_4	s_3	c_3	B_3	A_3	$prop_3$	gen_3	87777888F8888000

Table 3-6 The expression of LUTs in AFM2

LUT	INIT value (Hex)	Expression
LUT1	6AC06AC0A0A0A0A0	$P_0 = A_0B_0$ $P_1 = (A_1B_0) \oplus (A_0B_1)$
LUT2	1E665AAAB4CCF000	$P_2 = (A_1A_0B_1B_0) \oplus (A_2B_0) \oplus (A_1B_1) \oplus (A_0B_2)$
LUT3	6AC06AC080008000	$c_2 = A_3B_1A_2B_0$ $s_1 = (A_3B_0) \oplus (A_2B_1)$
LUT4	953F6AC02A008000	$prop_0 = (s_1 \oplus (A_1 \oplus B_2)) \oplus (A_0B_3)$ $gen_0 = (s_1 \oplus (A_1B_2))(A_0B_3)$
LUT5	96665AAA3CCCF000	$s_2 = (A_3B_1) \oplus (A_2B_2) \oplus (A_1B_3)$
LUT6	807F7F8000808000	$prop_1 = (s_2 \oplus c_2) \oplus (s_1A_1B_2)$ $gen_1 = (s_2 \oplus c_2)(s_1A_1B_2)$
LUT7	E888A000C0000000	$c_3 = A_3A_2B_2B_1 + A_3A_1B_3B_1 + A_2A_1B_3B_2$
LUT8	800080006AC06AC0	$s_3 = (A_3B_2) \oplus (A_2B_3)$ $c_4 = A_3B_2A_2B_3$
LUT9	936C936C20802080	$prop_2 = (s_3 \oplus c_3) \oplus (s_2c_2)$ $gen_2 = (s_3 \oplus c_3)(s_2c_2)$
LUT10	87777888F8888000	$prop_3 = (s_3c_3) \oplus (A_3B_3) \oplus c_4$ $gen_3 = (s_3c_3) \oplus (A_3B_3)c_4 + s_3c_3A_3B_3$

Table 3-5 shows the input and output configurations for each LUT in AFM2, along with the INIT value for each LUT. The total area of AFM2 is 10 LUTs and the critical path involves 2 LUTs and a carry chain. Table 3-6 illustrates the expressions of each LUT in AFM2.

3.3.4 Approximate 4×4 Multiplier 3 (AFM3)

Although the carry chain is fast to perform the arithmetic function, it still causes larger hardware consumptions than stand-alone LUT. Therefore, to further improve the hardware performance of approximate multiplier, approximate 4×4 multiplier 3 (AFM3) is proposed by using only LUTs.

Figure 3-9 shows the structure of AFM3, which does not include the carry chain. In each LUT, the inexact carry result from preceding column is computed inexactly as the result of AND operation as shown in the shadow part in Figure 3-9. For example,

column 2 consists of elements B_2, B_1, B_0, A_2, A_1 and A_0. Five AND gates in shadow part in LUT3 compute the inexact carry result from column 2. In AFM3, the multiplication results of P_0, P_1,..., P_7 are computed in parallel and the critical path is shortened to 2 LUTs. The input and output configurations and INIT values are shown in Table 3-7. Table 3-8 shows the expressions of AFM3, which is simpler than previous two designs.

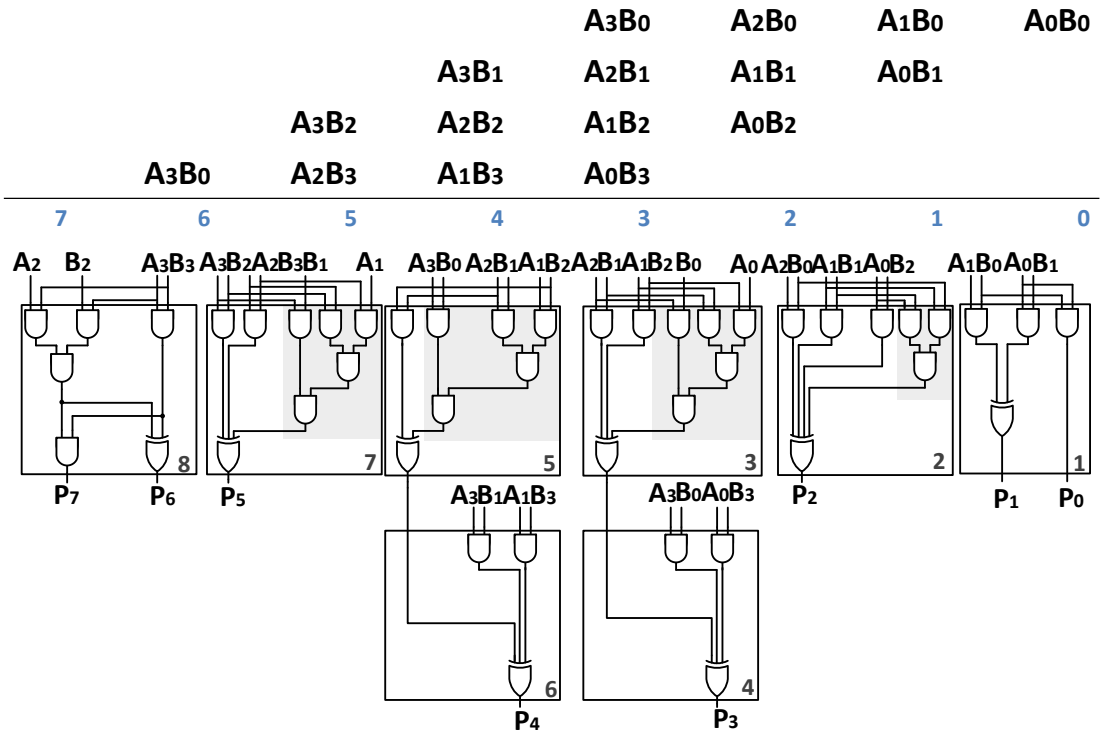


Figure 3-9 The structure of AFM3. Eight LUTs are used to produce the results in parallel.

Table 3-7 Input and output configurations for each LUT in AFM3.

LUT	Input configuration						Output configuration		INIT value (Hex)
	I5	I4	I3	I2	I1	I0	O6	O5	
LUT1	1	1	B_1	B_0	A_1	A_0	P_1	P_0	6AC06ACA0A0A0A0
LUT2	B_2	B_1	B_0	A_2	A_1	A_0	P_2		1E665AAAB4CCF000
LUT3	B_2	B_1	B_0	A_2	A_1	A_0	*temp1*		94B46CCCF0F00000
LUT4	1	*temp1*	B_3	B_0	A_3	A_0	P_3		953F6AC0953F6AC0
LUT5	B_2	B_1	B_0	A_3	A_2	A_1	*temp2*		64446CCC00000000
LUT6	1	*temp2*	B_3	B_1	A_3	A_1	P_4		953F6AC0953F6AC0
LUT7	B_3	B_2	B_1	A_3	A_2	A_1	P_5		94B46CCCF0F00000
LUT8	1	1	B_3	B_2	A_3	A_2	P_7	P_6	800080004C004C00

Table 3-8 The expression of LUTs in AFM3.

LUT	INIT value (Hex)	Expression
LUT1	6AC06ACA0A0A0A0	$P_0 = A_0B_0$ $P_1 = (A_1B_0) \oplus (A_0B_1)$
LUT2	1E665AAAB4CCF000	$P_2 = ((A_1B_0A_0B_1) \oplus A_2B_0) \oplus (A_1B_1) \oplus (A_0B_2)$
LUT3	94B46CCCF0F00000	$temp1 = ((A_2B_0 \mid A_1B_1)A_0B_2) \oplus (A_2B_1) \oplus (A_1B_2)$
LUT4	953F6AC0953F6AC0	$P_3 = (A_3B_0) \oplus (A_0B_3) \oplus temp1$
LUT5	64446CCC00000000	$Temp2 = ((A_3B_0 \mid A_2B_1)A_1B_2) \oplus (A_2B_2)$
LUT6	953F6AC0953F6AC0	$P_4 = (A_3B_1) \oplus (A_1B_3) \oplus temp2$
LUT7	94B46CCCF0F00000	$P_5 = ((A_3B_1 \mid A_2B_2)A_1B_3) \oplus (A_3B_2) \oplus (A_2B_3)$
LUT8	800080004C004C00	$P_6 = (A_2A_3)(A_1B_1)(A_2B_1)$ $P_7 = (A_3B_2 \mid A_2B_3) \oplus (A_2B_1)$

Therefore, the dot diagrams of three types of proposed 4×4 multipliers can be expressed in the Figure 3-10. AFM1 restricts the carry from column 2 to 3, and AFM2 is optimized on AFM1. AFM3 processes the elements in parallel with eight LUTs.

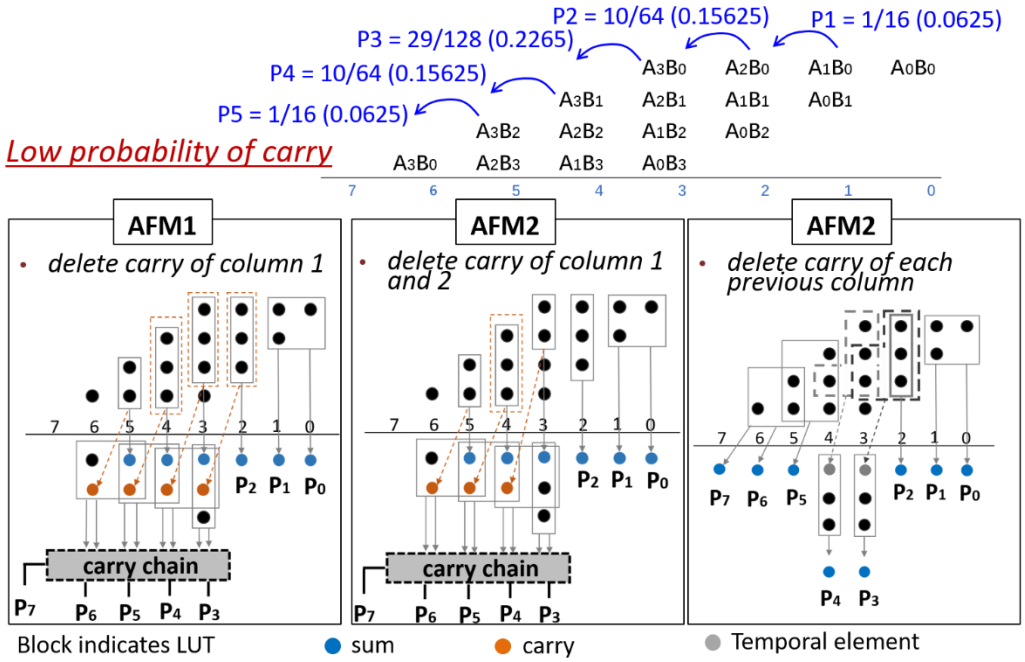


Figure 3-10 The dot diagrams of three types of proposed 4×4 multipliers.

3.4 Approximate Large Multipliers using Proposed Approximate 4×4 Multipliers as Elementary Modules

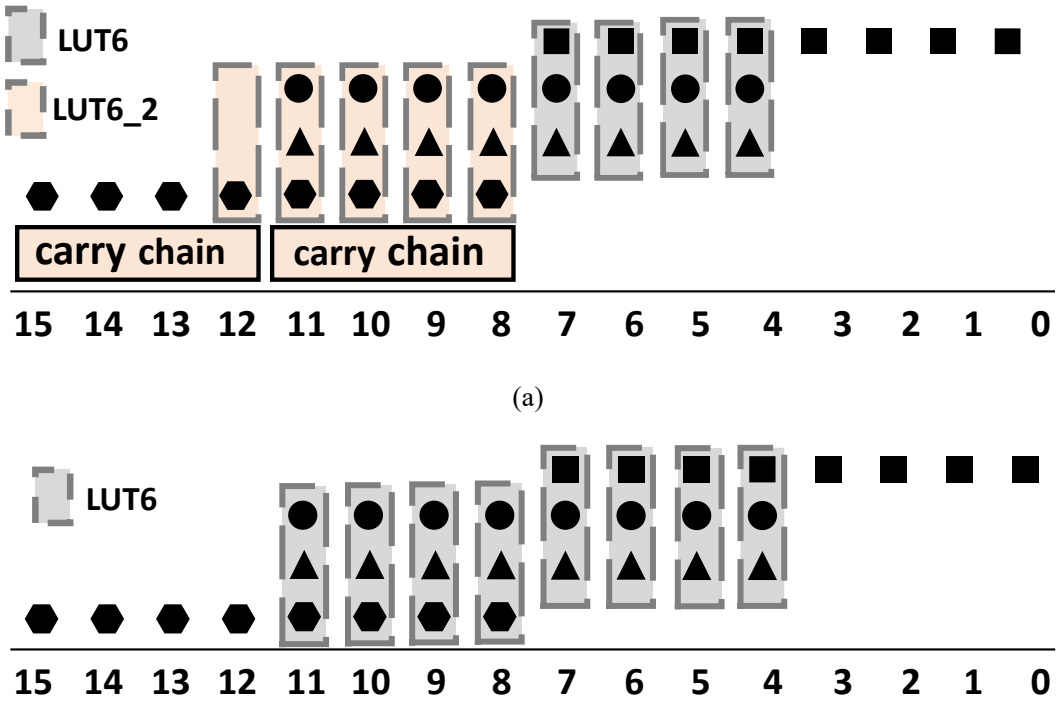
Recall the approach of architectural-space construction to construct large multiplier, as introduced in Eq. 2-1 (Section 2.2.1). The proposed approximate 4×4 multipliers are regarded as the elementary modules to construct larger multiplier. In this work, 8×8 multiplier design is built from 4×4 multipliers on architectural space. It is worth mentioning that, this approach is amenable to other size multipliers, it means the proposed 4×4 multiplier can further construct larger multiplier, such as 16×16 and 32×32 multiplier. It provides the possibility that the proposed methodology with architectural-space construction could be extended to larger multiplier.

Firstly, four 8-bit products are generated from four 4×4 multipliers (i.e. $A_L \times B_L$, $A_L \times B_H$, $A_H \times B_L$, $A_H \times B_H$). Then, the adder is used to sum four 8-bit products. Generally, the exact adder for summing four 8-bit products consists of nine LUTs and three carry chains [47]. The hardware consumptions of exact adder, especially delay consumption, are large because of the serial carry propagation path. To produce the final 8×8 multiplication product with low cost, we propose two inexact adders to compute the result.

1) Inexact adder 1 (IA1)

Figure 3-11 (a) shows inexact adder 1 (IA1) which is proposed based on the column-significance. Columns 4~7 are positioned at low part of the 8×8 multiplier, and the highest weight of this part is 2^7. The significance of this part to overall multiplication result is low. Therefore, the results of these columns are inexactly accumulated by cutting the carry propagation of adjacent columns. OR operation is usually the most appropriate choice for designing inexact adder. This is because the result is only one binary number when the carry is cut, and OR operation could produce the result as '1' when one element among inputs is '1'. Therefore, in IA1, 4 LUTs

configured with OR operation are used on columns 4~7, while the exact adder is used on columns 8~15.



(a)

(b)

Figure 3-11 Two proposed inexact adders. The dots of ■, ●, ▲ and ⬣ indicates the products from $A_L \times B_L$, $A_L \times B_H$, $A_H \times B_L$, and $A_H \times B_H$, respectively. (a) IA1: inexact operation is used on columns 4~7, while exact operation is used on columns 8~15. (b) IA2: eight LUTs are used to produce the results in parallel.

2) Inexact adder 2 (IA2)

Based on IA1, a highly-inexact adder is proposed to further reduce hardware consumptions. Inexact adder 2 (IA2) is shown in Figure 3-11 (b), where eight LUTs are used to produce the results in parallel. The carry propagation among columns is cut by the LUTs. The logic function for each LUT is OR operation and the INIT value is 'FEFEFEFEFEFEFEFE' (hex). The latency of the adder is efficiently reduced by producing the results in parallel.

3.5 Experiment Results and Discussion

In this section, 4×4 multipliers are firstly evaluated, followed by the evaluation of 8×8 multipliers. Finally, approximate designs are assessed on image sharpening processing.

3.5.1 Experiment Setup

To clarify the contributions of the proposed multipliers, the proposed FPGA-based approximate multiplier design was implemented and compared with the default exact multiplier, Xilinx multiplier IP [69], and approximate multipliers in [55] (UDM), [63] (C3), [46] (SMA), [47] (Cc, Ca). Among them, UDM and C3 are ASIC-based approximate multipliers, while SMA, Cc and Ca are the state-of-the-art FPGA-based approximate multipliers. In [47], the approaches for 8-bit Ca, Cc both are extended from the 4-bit approximate multiplier (Ca). The difference in 8-bit Ca and Ca is on the adder. Therefore, the evaluation of 4-bit approximate multiplier of [47] is on Ca.

For accuracy analysis, approximate multipliers were evaluated in terms of mean error distance (MED), mean relative error distance (MRED) and error rate (ER). These three metrics have been defined in Section 2.4.2. The functional models of proposed multipliers were implemented using Matlab and an exhaustive simulation (i.e. 256 patterns for 4×4 multiplier and 65536 patterns for 8×8 multiplier) was performed for all approximate multipliers.

For evaluation of hardware performance, the proposed design was coded in Verilog. Then, the design was synthesized and implemented using Xilinx Vivado 18.3 for XC7VX330T device of Virtex-7 family. We implemented approximate multipliers C3, SMA, Cc and Ca using the open-source codes provided by [63], [46] and [47], respectively. UDM is one of variants in [63], hence we implemented it according to the open-source codes provided by [63]. All multipliers were synthesized and implemented in the same environment with default options. To precisely evaluate power, switching activity interchange format (.saif) file was captured during post place and route

functional simulation then the file was used to report power consumptions.

3.5.2 Evaluation of 4×4 Multipliers

Table 3-9 shows the accuracy comparison of approximate 4×4 multipliers. The proposed multiplier AFM1 achieves the lowest accuracy loss in terms of MED, MRED and ER. This is because the approximation is performed on only two columns in AFM1, while the approximation is applied on all columns in other deigns.

Table 3-9 Accuracy comparison of 4×4 multipliers.

Designs	MED	MRED (%)	ER (%)
AFM1	0.13	0.14	1.56
AFM2	1.50	2.94	17.19
AFM3	11.25	13.53	32.81
UDM [55]	3.13	2.61	19.14
C3 [63]	4.69	13.97	46.48
SMA [46]	10.75	12.62	35.94
Ca [47]	0.19	0.24	2.34

The hardware performance is shown in Table 3-10. The proposed AFM3 has the shortest latency and the smallest PDP, because it produces the final product in parallel. AFM2 is optimized on AFM1, while the latency of AFM2 is larger than that of AFM1. We found the reason from placed and routed schematic, which might cause the latency increase. In general, the total delay of a circuit is the summation of logic delay and net delay. The logic delays of AFM1 and AFM2 are almost same, because they both have one LUT6, one LUT6_2 and one carry chain on critical path. However, the net delay of AFM2 is larger than that of AFM1, because the connection of two LUTs in AFM2 is longer than that in AFM1. In other word, the actual hardware delay of routed interconnect in AFM2 is larger than that in AFM1. Therefore, AFM2 has a larger total delay than AFM1.

Table 3-10 Area, latency, power and PDP of 4×4 multipliers.

Designs	Area [LUTs]	Delay (ns)	Power (W)	PDP (nJ)
Exact	16	5.771	0.248	1.431
Xilinx Multiplier IP	15	5.975	0.245	1.464
AFM1	12	5.529	0.242	1.338
AFM2	10	5.880	0.240	1.411
AFM3	8	4.870	0.230	1.120
UDM [55]	13	5.683	0.235	1.336
C3 [63]	15	6.032	0.227	1.369
SMA [46]	7	5.174	0.229	1.185
Ca [47]	12	5.783	0.243	1.405

Combining with the results in Table 3-9 and 3-10, it is can be observed that the proposed 4×4 design outperforms other approximate multipliers with comparable accuracy. For example, both AFM1 and Ca have the MREDs which are less than 1%. However, the latency, power and PDP consumptions of AFM1 are lower than those of Ca. The same observation can be found for AFM2 with UDM, AFM3 with C3.

3.5.3 Evaluation of 8×8 Multipliers

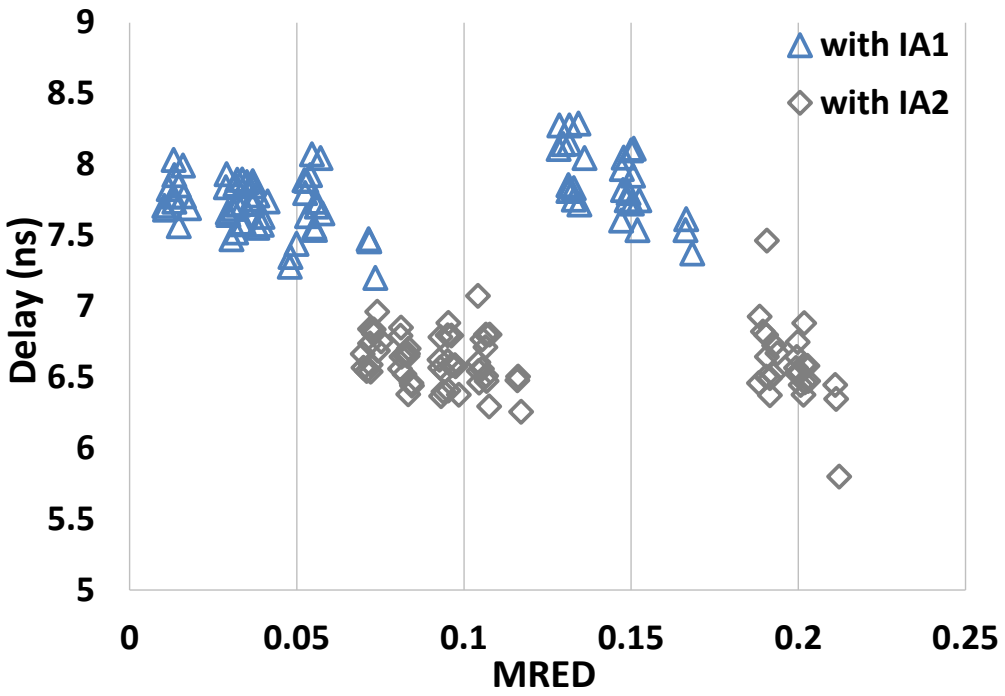
In this subsection, first all configurations for 8×8 multiplier are discussed which is constructed from proposed 4×4 multipliers and proposed inexact adders. Then, eight configurations are selected and implemented to evaluate the accuracy loss and hardware performance.

1) Latency and area evaluation to select configurations for the proposed design

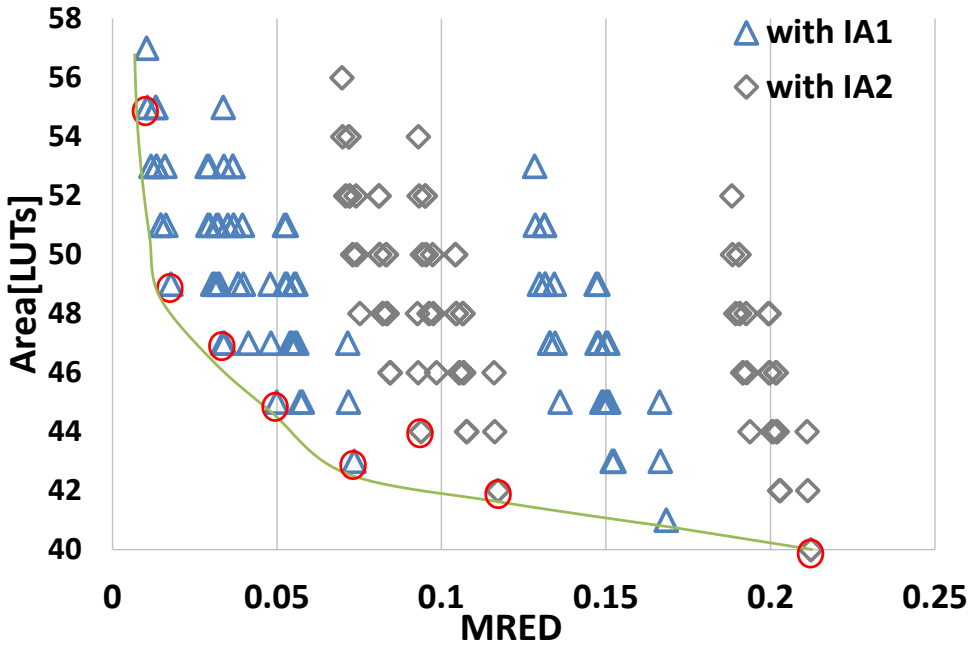
Figure 3-11 shows the latency and area of all configurations (i.e. 162 cases) for the 8×8 multiplier, with respect to MRED. The x-axis indicates the MRED value of each configuration and the y-axis indicates the latency in Figure 3-12 (a) and area in Figure 3-12 (b), respectively.

Different accuracy-hardware results can be achieved by different configurations. The tendency of area-MRED is more obvious than that of latency-MRED. Therefore, based on the area-MRED tendency shown in Figure 3-12 (b), select eight configurations

are selected for the proposed 8×8 multiplier. The green line indicates the best area-



(a)



(b)

Figure 3-12 The delay and area of all configurations for the proposed 8×8 multiplier. (a) Delay vs. MRED. (b) Area vs. MRED. The dots on the green line have the best area-MRED tradeoff, and the dots with red circles are selected.

MRED tradeoff. Five dots are selected for configurations with IA1, and three dots are selected for configurations with IA2. Table 3-11 illustrates the eight selected configurations, where T1 to T8 correspond to eight dots with red circles from left to right in Figure 3-12 (b).

Table 3-11 Configurations for the proposed 8×8 multipliers.

Designs	**Configuration**				
	$A_H \times B_H$	$A_H \times B_L$	$A_L \times B_H$	$A_L \times B_L$	adder
T1	AFM1	AFM1	AFM1	AFM2	IA1
T2	AFM1	AFM2	AFM2	AFM3	IA1
T3	AFM1	AFM2	AFM3	AFM3	IA1
T4	AFM1	AFM3	AFM3	AFM3	IA1
T5	AFM2	AFM3	AFM3	AFM3	IA1
T6	AFM1	AFM3	AFM3	AFM3	IA2
T7	AFM2	AFM3	AFM3	AFM3	IA2
T8	AFM3	AFM3	AFM3	AFM3	IA2

2) Accuracy analysis and hardware evaluation

Figure 3-13 shows the accuracy comparison of approximate 8×8 multipliers. The proposed 8×8 multipliers have the wide-range of accuracy, which provide the several choices for applications with different requirements. In terms of MED and MRED, T1 ranks second among all multipliers. Ca has the lowest error among all designs, because Ca employs the exact adder in the multiplier.

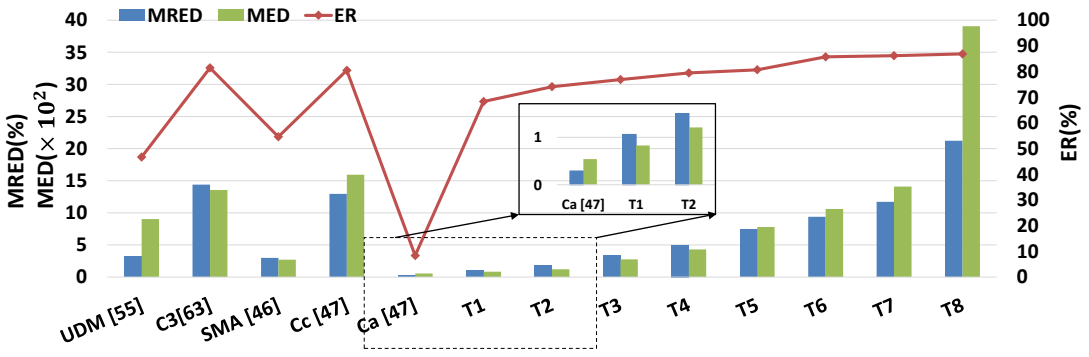
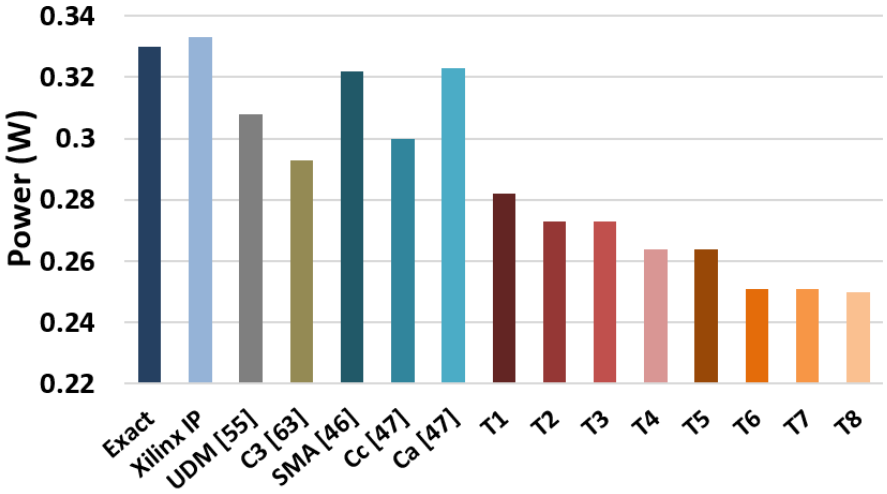
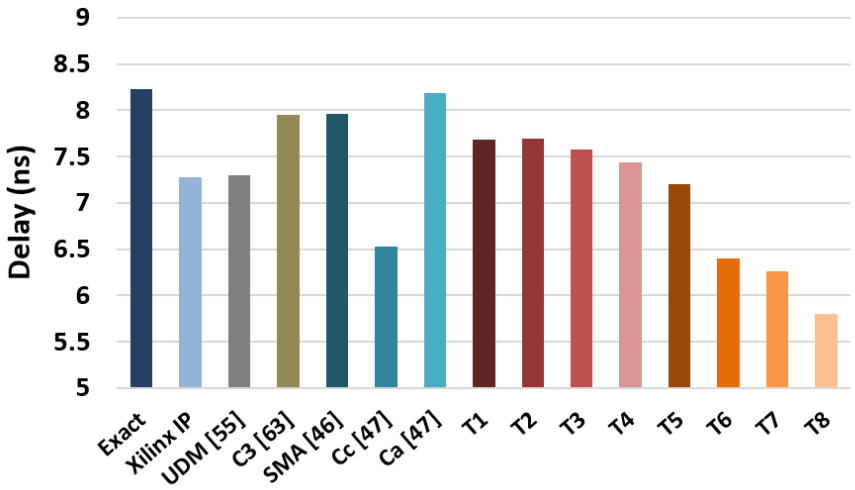


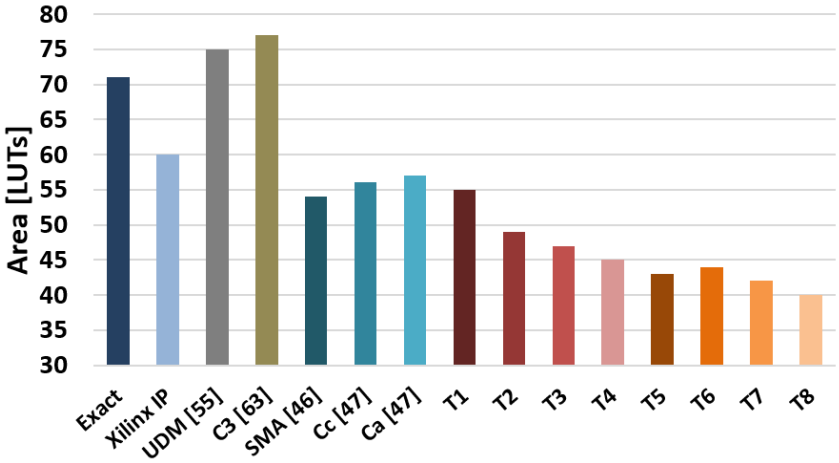
Figure 3-13 Accuracy comparison for approximate multipliers.



(a)



(b)



(c)

Figure 3-14 Hardware performance of the exact 8×8 multipliers and approximate 8×8 multipliers. (a) Power (b) Latency (c) Area.

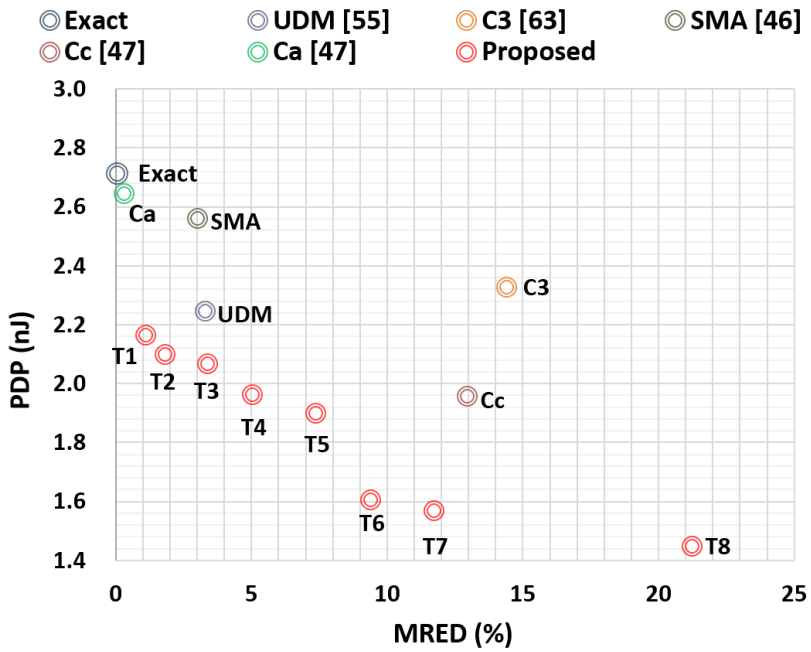


Figure 3-15 MRED and PDP for 8×8 multipliers.

The proposed 8×8 multipliers with selected configurations were synthesized and implemented under the same condition as Section 3.5.1. Exact multipliers and other previous approximate multipliers were evaluated under the same environment as the proposed design. Figure 3-14 shows the power, delay and area of all multipliers with 8-bit input.

The proposed multiplier T8 achieves the lowest power, latency and area among all multipliers. Compared with the exact multiplier, T8 has the power saving of 24.24%, latency saving of 29.50% and area saving of 43.66%. UDM and C3 are both ASIC-based designs and their LUTs on FPGA-based implementation are even larger than the area of the exact multiplier. The proposed design achieves more hardware improvements than SMA and Ca, which are FPGA-based approximate multipliers. For these improvements by the proposed multipliers, we can consider the construction of multipliers to explain. In SMA and Ca, exact adder is used to sum four partial products from four 4×4 multipliers. Exact adder consists of several carry chains and the serial carry propagation path, which leads to small area yet high power consumption. In contrast, in the proposed 8×8 multiplier design, the inexact adder is used to sum four 8-bit products. It significantly improves hardware performance, especially power and

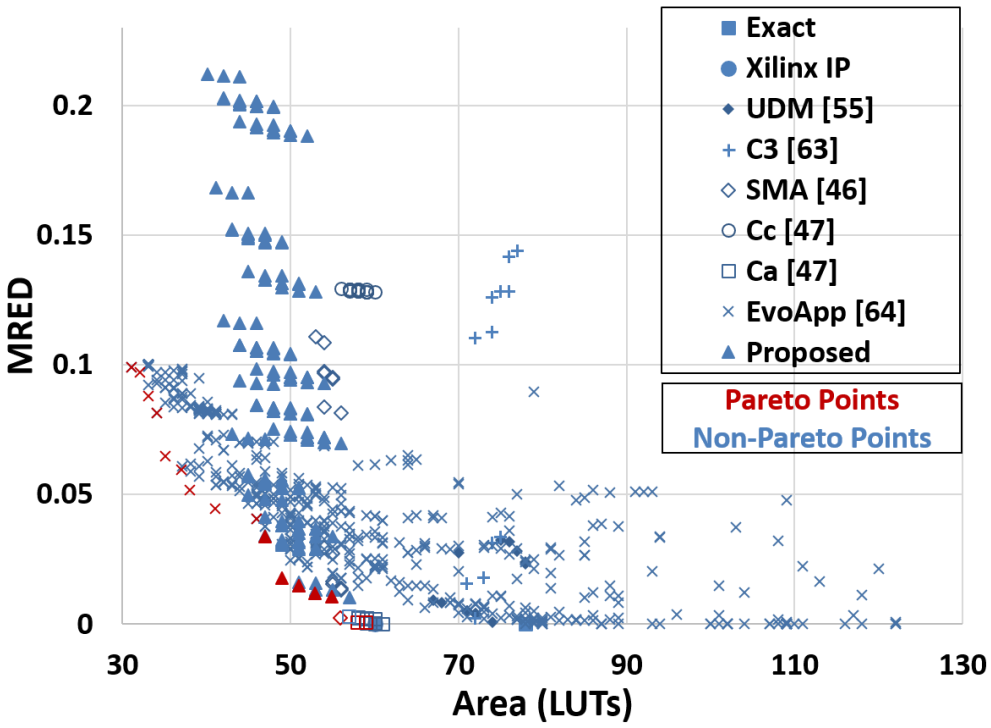
delay efficiency.

An intuitive comparison of accuracy-hardware tradeoff in terms of MRED and PDP for all approximate multipliers is shown in Figure 3-15. The proposed design has the lowest PDP on the same MRED, and the smallest MRED on the same PDP. For example, the MREDs of T3 and SMA are around 0.03. The PDP of T3 is 2.07nJ, while SMA has PDP of 2.56nJ. Overall, the proposed design has a better MRED-PDP result than other approximate designs.

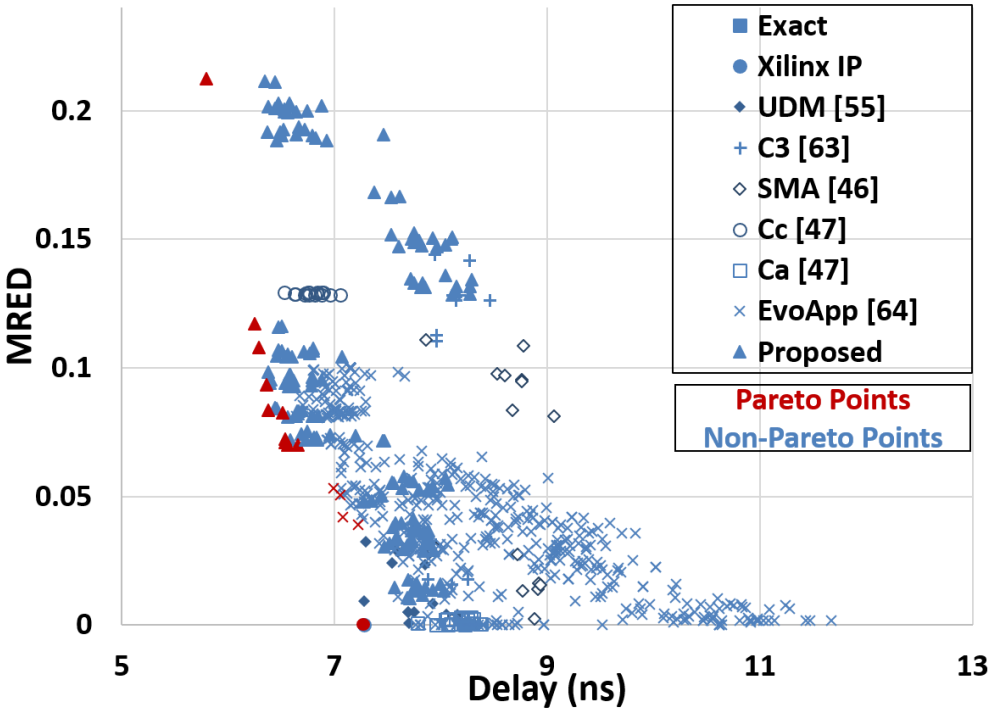
3) Pareto-optimal analysis

Pareto-optimal analysis is a statistical technique to search the best results in terms of two related metrics. Figure 3-16 compares all possible configurations of the proposed 8×8 multiplier, exact multipliers and other approximate multipliers in [55] (UDM), [63] (C3), [46] (SMA), [47] (Cc, Ca) and [64] (EvoApp). All configurations were evaluated under the same condition as Section 3.5.1. In Figure 3-16, y-axis indicates the MRED, and x-axis indicates the area result in Figure 3-16 (a) and the latency result in Figure 3-16 (b), respectively.

The pareto optimal analysis reveals that the EvoApp design has the smaller area than other designs for MRED on the range of 0.05~0.1. However, when MRED is lower than 0.05, the proposed design requires less LUTs. For MRED-latency result shown in Figure 3-16 (b), the proposed design has more pareto points than other designs. Actually, in the proposed design, all non-dominated configurations are the configurations with the proposed adder IA2 in which eight LUTs are processed in parallel. This causes that EvoApp requires smaller area than the proposed design, while the proposed design costs shorter delay.



(a)



(b)

Figure 3-16 Pareto optimal analysis for the 8×8 multipliers. (a) MRED vs. delay. (b) MRED vs. area.

3.5.4 Image Processing Application

(a) exact (b) T1 (c) T2

(d) T3 (e) T4 (f) T5

(g) T6 (h) T7 (i) T8

Figure 3-17 Processed images by exact multiplier and proposed multipliers.

To test approximate multipliers on application, image sharpening algorithm [58] is selected, because it is widely utilized to evaluate approximate multipliers. The introduction and function of this application have been introduced in Section 2.4.4. The peak signal-to-noise ratio (PSNR) is a metric to assess the quality of processed image compared with the exact image and defined in [42]. The input image for this application is 512×512 grayscale bitmap image with 8-bit pixels. Structural similarity index (SSIM) is another metric to measure the quality of processed image and we used the Matlab function $ssim$ to calculate it.

Figure 3-17 shows the processed images from the exact multiplier and proposed approximate multipliers. The difference is imperceptible among the images processed

by exact multiplier and the proposed multipliers of T1~T3. Figure 3-18 shows the PSNR and SSIM results of all approximate multipliers. The SSIM results of T1~T3 are higher than 99.0%. T8 is the worst case in the proposed design which is still sufficiently exact for error-tolerant applications, because the PSNR of 20dB can be regarded as acceptable [72].

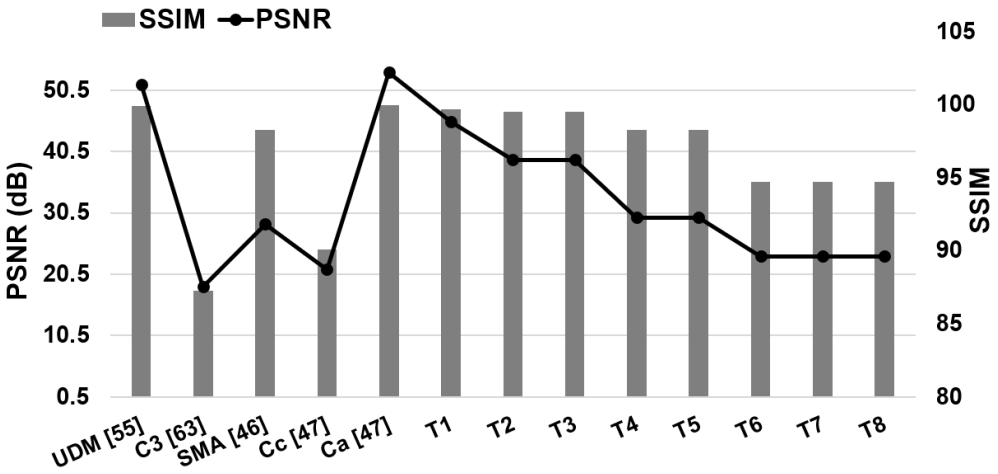


Figure 3-18 PSNR and SSIM values of processed images by approximate multipliers.

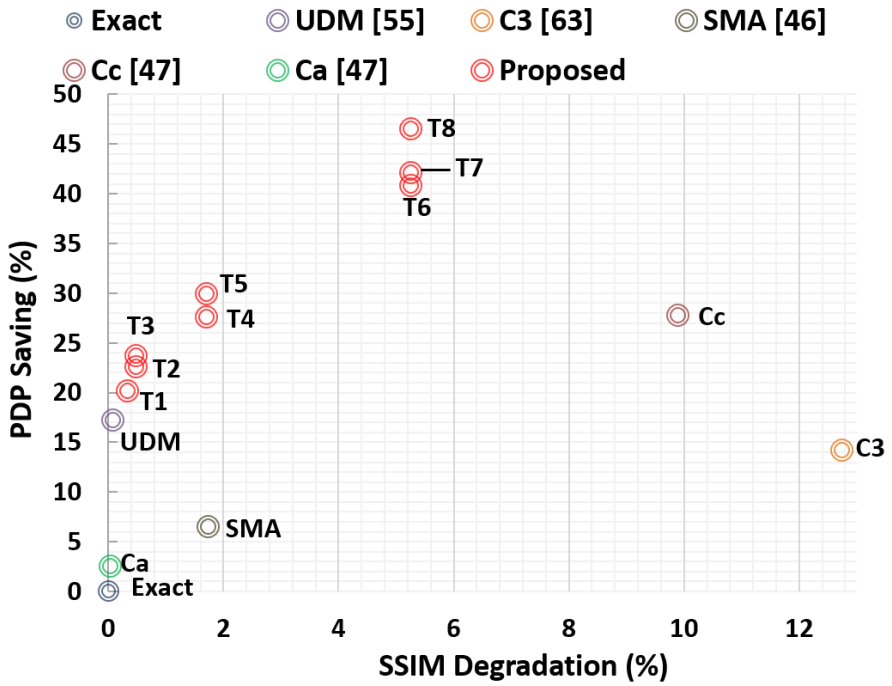


Figure 3-19 SSIM degradation and PDP saving of all multipliers.

The SSIM degradation and PDP saving achieved by all approximate multipliers are shown in Figure 3-19. On the same range of SSIM degradation, the proposed design has the higher PDP saving than other designs. Overall, T3 among the proposed multipliers is recommended for error-tolerant applications.

3.6 Summary

In this chapter, approximate FPGA-based 8×8 multiplier design is proposed to achieve lower hardware consumptions than exact multiplier. Firstly, this work proposes approximate 4×4 multipliers with LUTs and carry chain. The approximation is mainly on the carry propagation path. Then, the design of 8×8 multipliers is explored by using proposed 4×4 multipliers as elementary modules. For the adder summing the products from small multipliers, two types of inexact adders are proposed. This first adder approximates the result on low columns, while the second adder produces the product in parallel. The critical path is shortened by inexact adders. A wide-range of approximate 8×8 multipliers is provided for applications with different accuracy-hardware requirements. The experimental results demonstrate that the proposed multiplier design can significantly reduce the hardware consumptions. Compared with exact multiplier, 20.18%~46.59% PDP saving and 22.54%~43.66% area savings can be achieved by the proposed design. Compared with the state-of-the-art FPGA-based multiplier [47], the proposed delivers more 18.07% PDP saving under the similar error (MRED) of 1%; compared with [46], more 19.24% PDP savings can be achieved by the proposed design with the similar MRED of 3%.

4. Conclusion and Future Work

4.1 Conclusion

This dissertation targets at the approximate multipliers for error-tolerant applications. The proposals focus on probability-driven carry-restricted compressors for ASIC-based approximate multiplier and FPGA-based approximate multiplier. They both can achieve considerable trade-off between accuracy loss and hardware saving.

Chapter 2 presents the ASIC-based approximate multiplier. There are two contributions. Firstly, probability-driven inexact compressors are proposed by analyzing the probability distribution of partial products. The carry of this compressor is restricted (converted) to the same position with the sum. This compressor design does not include XOR gates, hence it is easily synthesized to area- and energy-efficient cells. More importantly, this is first attempt to introduce a unified expression and extend the input-width of inexact compressor up to 8-bit, which reduces the accumulation stage. By utilizing the proposed compressors, the approximate multiplier achieves more 26.28% area saving, 25.48% power saving and 16.39% delay saving, compared with multiplier without the proposed compressors. The second contribution is a grouped error-recovery scheme which is proposed to compensate error. This error recovery method is a derivate of probability-driven compressor, and the conventional adder can be optimized. The operation in this method is in the form of group. Compared with the previous error-recovery approaches implemented by conventional adder in bit-wise, the proposed scheme employs a simplified adder and significantly shortens the critical path. In addition, this error recovery scheme provides five variants of the proposed multiplier to achieve different accuracy-hardware results. Compared with the exact multiplier, the most efficient variant reduces the area by 50.97%, power by 70.75% and delay by 52.42%. Compared with the previous approximate multiplier with error recovery, under the similar accuracy loss of 1% (in terms of MRED), the proposed design achieves more 50.63% PDP saving.

Chapter 3 proposes an FPGA-based approximate multiplier. There are two contributions. Firstly, this research carefully considers the structure of FPGA-fabric, whereas the most approximate multipliers focused on the ASICs. Three types of approximate 4×4 multipliers with different performance are proposed, which are elementary module for the proposed approximate 8×8 multiplier. In 4×4 multiplier, the partial products are accumulated with compressor implemented by LUTs. By considering the low probability of carry, the carry computation is deleted in the compressor. The approximation on carry result significantly reduces the delay consumption. As a result, compared with the exact multiplier, the proposed 4×4 multiplier saves area up to 8 LUTs and delay of 15.62%. The second contribution is that an 8×8 multiplier design is constructed from elementary modules and the proposed inexact adders. These two inexact adders are proposed to cut the carry propagation when summing the products from four small multipliers. More importantly, all possible combinations are discussed to provides multiplier choices for the reconfigurability of FPGAs. As a result, the Pareto-optimal analysis is discussed to demonstrate that the proposed design has a better accuracy-hardware performance than previous works. Compared with the exact multiplier, the proposed design can reduce area by 43.66% and power by 24.24%. The critical path latency reduction is up to 29.50%. Compared with the state-of-the-art FPGA-based multiplier [47], the proposed one achieves more 18.07% PDP saving when MRED is around 1%; compared with [46], more 19.24% PDP savings can be achieved by the proposed design with the similar MRED of 3%.

4.2 Future Work

In this research, 8×8 integer approximate multipliers have been studied. There are three future works need to be considered. Firstly, the proposed approach is easily to be extended to signed integer number multipliers and floating-point number multipliers. In this dissertation, the method of the extension to signed integer number multiplier is discussed. To further show the potential of the approximate signed multiplier, more applications such as CNNs should be considered to test. Recently, to effectively balance

the huge computations in CNNs and limited hardware resource, floating-point numbers in networks are replaced by signed integer numbers. It provides the opportunity to apply approximate signed multipliers. To fully provide the choices for wide-range applications, approximate floating-point multipliers also need to be considered. In floating-point number multiplier, the mantissa part is the integer multiplication, which could use the proposed unsigned integer multiplier. Secondly, developing a construction method of higher order approximate multipliers such as 16-bit or 32-bit based on the proposed idea is another future work. The current research work is easily extended to 16-bit with four 8-bit multipliers. Thirdly, more applications will be considered to test the quality of the proposed method. Especially, this dissertation discussed approximate FPGA-based multiplier and FPGA has becoming a promising platform to accelerate a lot of data. Therefore, more applications using FPGA need to be considered.

Reference

[1] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, “Analysis and characterization of inherent application resilience for approximate computing,” in *Proceedings of the 50th Annual Design Automation Conference (DAC)*, pp. 1-9, May 2013.

[2] J. Han and M. Orshansky, “Approximate computing: An emerging paradigm for energy-efficient design,” in *Proceedings of the IEEE European Test Symposium (ETS)*, pp. 1-6, May 2013.

[3] G. E. Moore, “Cramming more components onto integrated circuits”, *Proc. IEEE*, vol. 86, no. 1, pp. 82-85, Jan. 1998.

[4] R. H. Dennard, F. H. Gaensslen, H. N. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, “Design of ion-implanted MOSFET's with very small physical dimensions,” *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 9, no. 5, pp. 256-268, 1974.

[5] Q. Xu, T. Mytkowicz, and N. S. Kim, “Approximate computing: A survey,” *IEEE Design & Test*, vol. 33, no. 1, pp. 8-22, Feb. 2016

[6] S. Mittal, “A survey of techniques for approximate computing,” *ACM Computing Surveys*, Article No. 62, pp. 1-33, Mar. 2016.

[7] A. Agrawal, J. Choi, K. Gopalakrishnan, S. Gupta, R. Nair, J. Oh, D. A. Prener, S. Shukla, V. Srinivasan, and Z. Sura, “Approximate computing: Challenges and opportunities,” in *IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1-8, Oct. 2016.

[8] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, “Approximate computing and the quest for computing efficiency,” in *52nd ACM/EDAC/IEEE Design*

Automation Conference (DAC), pp. 1-6, July 2015.

[9] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "ApproxANN: An approximate computing framework for artificial neural network," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 701-706, Mar. 2015.

[10] V. K. Chippa, H. Jayakumar, D. Mohapatra, K. Roy, and A. Raghunathan, "Energy-efficient recognition and mining processor using scalable effort design," in *Proceedings of the IEEE 2013 Custom Integrated Circuits Conference (CICC)*, pp. 1-4, Sept. 2013.

[11] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard, "Quality of service profiling," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pp. 25-34, May 2010.

[12] J. Meng, S. Chakradhar, and A. Raghunathan, "Best-effort parallel execution framework for recognition and mining applications," in *IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pp. 1-12, May 2009.

[13] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," *ACM Transactions on Computer Systems (TOCS)*, vol.32, no.3, pp.1-23, Sept. 2014.

[14] A. Yazdanbakhsh, G. Pekhimenko, B. Thwaites, H. Esmaeilzadeh, O. Mutlu, and T. C. Mowry, "RFVP: Rollback-free value prediction with safe-to-approximate loads," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol.12 no.4, pp.1-26, Jan. 2016.

[15] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," in *46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1-12, Dec. 2013.

[16] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, “Neural acceleration for general-purpose approximate programs,” in *45th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 449-460, Dec. 2012.

[17] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, “Architecture support for disciplined approximate programming,” in *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, pp. 301-312, Mar. 2012.

[18] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, “IMPACT: imprecise adders for low-power approximate computing,” in *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 409-414, Aug. 2011.

[19] C. Liu, J. Han, and F. Lombardi, “A low-power, high-performance approximate multiplier with configurable partial error recovery,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1-4, Mar. 2014.

[20] V. Leon, G. Zervakis, D. Soudris and K. Pekmestzi, “Approximate Hybrid High Radix Encoding for Energy-Efficient Inexact Multipliers,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 3, pp. 421-430, Mar. 2018.

[21] H. Jiang, F. Lombardi, and J. Han, “Low-Power Unsigned Divider and Square Root Circuit Designs Using Adaptive Approximation,” *IEEE Transactions on Computers (TC)*, vol. 68. No.11, pp.1635-1646, May 2019.

[22] D. Shin, and S. K. Gupta, “Approximate logic synthesis for error tolerant applications,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE),* pp. 957-960, Mar. 2010.

[23] J. Miao, A. Gerstlauer, and M. Orshansky, “Approximate logic synthesis under general error magnitude and frequency constraints,” in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 779-786, Nov. 2013.

[24] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, “Design of voltage-scalable meta-functions for approximate computing,” in *Design, Automation & Test in Europe (DATE)*, pp. 1-6, Mar. 2011.

[25] K. Palem, and A. Lingamneni, “Ten years of building broken chips: The physics and engineering of inexact computing,” *ACM Transactions on Embedded Computing Systems (TECS)*, no.87, pp. 1-23, May 2013.

[26] C. Y. Chen, J. Choi, K. Gopalakrishnan, V. Srinivasan, and S. Venkataramani, “Exploiting approximate computing for deep learning acceleration,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 821-826, Mar. 2018.

[27] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, “Deep learning with limited numerical precision,” in *International Conference on Machine Learning (ICML)*, pp. 1737-1746, June. 2015.

[28] N. P. Jouppi, et al., “In-datacenter performance analysis of a tensor processing unit,” in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1-12, June 2017.

[29] P. Düben, S. Yenugula, J. Augustine, K. Palem, J. Schlachter, C. Enz, and T. N. Palmer, “Opportunities for energy efficient computing: A study of inexact general purpose processors for high-performance and big-data applications,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 764-769, Mar. 2015.

[30] C. Guo, L. Zhang, X. Zhou, W. Qian, and C. Zhuo, “A Reconfigurable Approximate Multiplier for Quantized CNN Applications,” in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 235-240, Jan. 2020.

[31] M. Masadeh, O. Hasan, and S. Tahar, “Input-conscious approximate multiply-accumulate (MAC) unit for energy-efficiency,” *IEEE Access*, vol. 7, pp. 147129-147142, Oct. 2019.

[32] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, “Low-power digital signal processing using approximate adders,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 32, no.1, pp. 124-137, Dec. 2012.

[33] H. Jiang, J. Han, and F. Lombardi, “A comparative review and evaluation of approximate adders,” in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI (GLSVLSI)*, pp. 343-348, May 2015.

[34] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, “On reconfiguration-oriented approximate adder design and its application,” in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 48-54, Nov. 2013.

[35] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi, “Approximate XOR/XNOR-based adders for inexact computing,” in *13th IEEE International Conference on Nanotechnology (IEEE-NANO)*, pp. 690-693, Aug. 2013.

[36] H. Jiang, J. Han, F. Qiao, and F. Lombardi, “Approximate radix-8 booth multipliers for low-power and high-performance operation,” *IEEE Transactions on Computers (TC)*, vol. 65, no. 8, pp. 2638-2644, Oct. 2015.

[37] W. Liu, L. Qian, C. Wang, H. Jiang, J. Han, and F. Lombardi, “Design of approximate radix-4 booth multipliers for error-tolerant computing,” *IEEE Transactions on Computers (TC)*, vol. 66, no. 8, pp. 1435-1441, Feb. 2017.

[38] S. Hashemi, R. I. Bahar, and S. Reda, “DRUM: A dynamic range unbiased multiplier for approximate applications,” in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 418-425, Nov. 2015.

[39] I. Qiqieh, R. Shafik, G. Tarawneh, D. Sokolov, and A. Yakovlev, “Energy-efficient approximate multiplier design using bit significance-driven logic compression,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE),* pp. 7-12, Mar. 2017.

[40] M. Ha, and S. Lee, “Multipliers with approximate 4–2 compressors and error recovery modules,” *IEEE Embedded Systems Letters*, vol. 10, no. 1, pp. 6-9, Mar. 2018.

[41] M. S. Ansari, H. Jiang, B. F. Cockburn, and J. Han, “Low-power approximate multipliers using encoded partial products and approximate compressors,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS)*, vol. 8, no. 3, pp. 404-416, May 2018.

[42] A. Momeni, J. Han, and F. Lombardi, “Design and analysis of approximate compressors for multiplication,” *IEEE Transactions on Computers (TC)*, vol.64, no.4, pp.984–994, Apr. 2015.

[43] Z. Yang, J Han, and F. Lombardi, “Approximate compressors for error-resilient multiplier design,” in *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, pp. 183-186, Oct. 2015.

[44] D. Esposito, A. G. M. Strollo, E. Napoli, D. De Caro, and N. Petra, “Approximate multipliers based on new approximate compressors,” *IEEE Transactions on Circuits and Systems I: Regular Papers (TCAS-I)*, vol. 65, no. 12, pp. 4169-4182, June 2018.

[45] H. Jiang, C. Liu, F. Lombardi, and J. Han, “Low-power approximate unsigned multipliers with configurable error recovery,” *IEEE Transactions on Circuits and Systems I: Regular Papers (TCAS-I)*, vol. 66, no. 1, pp. 189-202, Jan. 2019.

[46] S. Ullah, S. S. Murthy, and A. Kumar, “SMApproxlib: library of FPGA-based approximate multipliers,” in *Proceedings of the 55th Annual Design Automation Conference (DAC)*, Article No. 157, pp. 1-6, June 2018.

[47] S. Ullah, S. Rehman, B. S. Prabakaran, F. Kriebel, M. A. Hanif, M. Shafique, and A. Kumar, “Area-optimized low-latency approximate multipliers for FPGA-based hardware accelerators,” in *Proceedings of the 55th Annual Design Automation Conference (DAC)*, Article No. 159, pp. 1-6, June 2018.

[48] S. Ullah, H. Schmidl, S. S. Sahoo, S. Rehman, and A. Kumar, “Area-optimized Accurate and Approximate Softcore Signed Multiplier Architectures,” *IEEE Transactions on Computers (TC)*, 2020. (Early Access)

[49] T. Yang, T. Ukezono, and T. Sato, “A low-power high-speed accuracy-controllable approximate multiplier design,” in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 605-610, Jan. 2018.

[50] S. Venkatachalam, and S. B. Ko, “Design of power and area efficient approximate multipliers,” *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 25, no. 5, pp. 1782-1786, May 2017.

[51] K. Y. Kyaw, W. L. Goh and K. S. Yeo, “Low-power high-speed multiplier for error-tolerant application,” in *IEEE International Conference of Electron Devices and Solid-State Circuits (EDSSC)*, pp. 1-4, Dec. 2010.

[52] T. Yang, T. Ukezono, and T. Sato, “Low-power and high-speed approximate multiplier design with a tree compressor,” in *IEEE International Conference on Computer Design (ICCD)*, pp. 89-96, Nov. 2017.

[53] P. Yadav, A. Pandey, K. R. Prasad, M. H. Vasantha, and Y. N. Kumar, “Low power approximate multipliers with truncated carry propagation for LSBs,” in *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 500-503, Aug. 2018.

[54] S. Boroumand, H. P. Afshar, P. Brisk, and S. Mohammadi, “Exploration of approximate multipliers design space using carry propagation free compressors,” in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 611-616, Jan. 2018.

[55] P. Kulkarni, P. Gupta, and M. Ercegovac, “Trading accuracy for power with an underdesigned multiplier architecture,” in *International Conference on VLSI Design (VLSID)*, pp. 346-351, Jan. 2011.

[56] C. H. Chang, J. Gu, and M. Zhang, “Ultra low-voltage low-power CMOS 4-2 and 5-2 compressors for fast arithmetic circuits,” *IEEE Transactions on Circuits and Systems I: Regular Papers (TCAS-I)*, vol. 51, no. 10, pp. 1985-1997, Oct. 2004.

[57] J. Liang, J. Han, and F. Lombardi, “New metrics for the reliability of approximate and probabilistic adders,” *IEEE Transactions on computers (TC)*, vol. 62, no. 9, pp. 1760-1771, June 2012.

[58] Synopsys, Power Compiler User Guide, Version D-2010.03-SP2, 2010.

[59] M. S. Lau, K. V. Ling, and Y. C. Chu, “Energy-aware probabilistic multiplier: design and analysis,” in *Proceedings of the international conference on Compilers, architecture, and synthesis for embedded systems (CASES)*, pp. 281-290, Oct. 2009.

[60] C. R. Baugh, and B. A. Wooley, “A two's complement parallel array multiplication algorithm,” *IEEE Transactions on computers (TC)*, vol. 100, no. 12, pp. 1045-1047, Dec. 1973.

[61] O. L. MacSorley, “High-speed arithmetic in binary computers,” *Proceedings of the IRE*, vol. 49, no. 1, pp. 67-91, Jan. 1961.

[62] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim, “Energy-Efficient Approximate Multiplication for Digital Signal Processing and Classification Applications,” *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 23, no. 6, pp. 1180-1184, June 2015.

[63] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, and J. Henkel, “Architectural-space exploration of approximate multipliers,” in *Proceedings of the IEEE/ACM International Conference on ComputerAided Design (ICCAD)*, pp.1-8, Jan. 2016.

[64] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, “EvoApproxSb: Library of approximate adders and multipliers for circuit design and benchmarking of

approximation methods," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 258-261, Mar. 2017.

[65] H. Parandeh-Afshar and P. Ienne, "Measuring and Reducing the Performance Gap between Embedded and Soft Multipliers on FPGAs," in *21st International Conference on Field Programmable Logic and Applications (FPL)*, pp. 225-231, Sept. 2011.

[66] N. Van Toan, and J. G. Lee, "FPGA-Based Multi-Level Approximate Multipliers for High-Performance Error-Resilient Applications," *IEEE Access*, vol. 8, pp. 25481-25497, Feb. 2020.

[67] Z. Ebrahimi, S. Ullah, and A. Kumar, "LeAp: Leading-one Detection-based Softcore Approximate Multipliers with Tunable Accuracy," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 605-610, Jan. 2020.

[68] I. Kuon, and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on computer-aided design of integrated circuits and systems (TCAD)*, vol. 26, no. 2, pp. 203-215, Jan. 2007.

[69] Intel, Integer Arithmetic IP Cores User Guide, https://www.altera.com /en_US/pdfs/literat ure/ug/ug_lpm_alt_mfug.pdf, 2017.

[70] Xilinx, LogiCORE IP Multiplier v11.2., https://www.xilinx.com/support/ documentation/ip_documentation/mult_gen_ds255.pdf, 2017.

[71] Xilinx, 7 Series FPGAs Configurable Logic Block User Guide, https:// www.xilinx.com/ support/documentation/user_guides/ ug474_7Series_CLB.pdf, 2016.

[72] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram, "RoBA multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 25, no. 2, pp.393-401, Feb. 2017.

[73] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600-612, 2004.

Publications

Journals:

[1] **Yi Guo**, Heming Sun, Ping Lei, and Shinji Kimura, "Approximate FPGA-Based Multipliers using Carry-Inexact Elementary Modules," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E103-A, No. 09, pp. 1054-1062, Sept. 2020.

[2] **Yi Guo**, Heming Sun, Ping Lei, and Shinji Kimura, "Design of Low-Cost Approximate Multipliers Based on Probability-Driven Inexact Compressors," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E102-A, No. 12, pp. 1781-1791, Dec. 2019.

International Conferences:

[1] Jie Li, **Yi Guo**, and Shinji Kimura, "Accuracy-Configurable Low-Power Approximate Floating-Point Multiplier Based on Mantissa Bit Segmentation", IEEE Region 10 Conference (TENCON), Osaka, Japan, pp. 1311-1316, Nov. 2020.

[2] **Yi Guo**, Heming Sun, and Shinji Kimura, "Small-Area and Low-Power FPGA-Based Multipliers using Approximate Elementary Modules," Asia and South Pacific Design Automation Conference (ASP-DAC), Beijing, China, pp. 599-604, Jan. 2020.

[3] Yufeng Xu, **Yi Guo**, and Shinji Kimura, "Approximate Multiplier Using Reordered 4–2 Compressor with OR-based Error Compensation," IEEE International Conference on ASIC (ASICON), Chongqing, China, pp. 1-4, Oct. 2019.

[4] **Yi Guo**, Heming Sun, and Shinji Kimura, "Energy-Efficient and High-Speed Approximate Signed Multipliers with Sign-Focused Compressors," IEEE International System-on-Chip Conference (SOCC), Singapore, pp. 330-335, Sept. 2019.

[5] Xiaoting Sun, **Yi Guo**, Zhenhao Liu, and Shinji Kimura, "A Radix-4 Partial Product Generation-Based Approximate Multiplier for High-speed and Low-power Digital Signal Processing," IEEE International Conference on Electronics, Circuits and Systems (ICECS), Bordeaux, France, pp. 777-780, Dec. 2018.

[6] **Yi Guo**, Heming Sun, Li Guo, and Shinji Kimura, "Low-Cost Approximate Multiplier Design using Probability-Driven Inexact Compressors," IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), Chengdu, China, pp. 291-294, Oct. 2018.

[7] **Yi Guo**, Heming Sun, and Shinji Kimura, "Design of Power and Area Efficient Lower-Part-OR Approximate Multiplier," IEEE Region 10 Conference (TENCON), Jeju, Korea, pp. 2110-2115, Oct. 2018.

[8] Zhenhao Liu, **Yi Guo**, Xiaoting Sun, and Shinji Kimura, "Energy-Efficient and High-Performance Approximate Multiplier Using Compressors Based on Input Reordering," IEEE Region 10 Conference (TENCON), Jeju, Korea, pp. 0545-0550, Oct. 2018.

Award:

[1] **Yi Guo**, Heming Sun, and Shinji Kimura, "Small-Area and Low-Power FPGA-Based Multipliers using Approximate Elementary Modules," *IEICE VLD Excellent Student Author Award for ASP-DAC 2020*, 2020.