Efficient architectures for computation of binary logarithm

Low, Joshua Yung Lih

2014

Low, J. Y. L. (2014). Efficient architectures for computation of binary logarithm. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/59971>

<https://doi.org/10.32657/10356/59971>

Efficient Architectures for Computation of Binary Logarithm

Joshua Low Yung Lih

School of Electrical and Electronic Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

June 2014

Acknowledgement

I would like to express my deepest gratitude to Associate Professor Jong Ching Chuen, my academic supervisor during my Ph.D. candidature, for his patient guidance and endless support. His liberal thinking towards creative and innovative ideas is the main motivation for me to look at the research problems from various prospective and hence uncover every bit of potential solutions. I am indebted to him for all his time and effort in reviewing this thesis and the manuscripts of our publications. Also, I sincerely appreciate his technical advice and constructive comments which lead me to the three newly proposed methods and their corresponding architectures presented in this thesis and many other ideas and directions for further research.

I would like to thank Associate Professor Chang Chip Hong for giving me the chances to participate numerous Chip's family technical meetings and presentations as well as social gatherings, which not only broaden my technical horizon but also enrich my research journey. I would also like to thank Associate Professor Siek Liter, Director of ViRTUS – IC Design Centre of Excellence, for providing an excellent research environment equipped with advance research resources for me to carry out my research work.

Special thanks to my friends in ViRTUS for their kind assistances and supports in solving various problems along my research works.

Table of Contents

Acknowledgement	i
Table of Contents	ii
Summary	v
List of Figures	vi
List of Tables	viii
List of Abbreviations	x
CHAPTER 1. Introduction	1
1.1 Background and Motivation.....	1
1.2 Research objectives.....	4
1.3 Major contributions.....	5
1.4 Organization of the thesis.....	7
CHAPTER 2. Literature Review.....	8
2.1 Mitchell-based Methods.....	8
2.1.1 Mitchell’s Straight Line Approximation	8
2.1.2 Combet’s Method.....	11
2.1.3 Hall’s Method	11
2.1.4 SanGregory’s Method	12
2.1.5 Abed’s Method.....	13
2.1.6 Kim’s Method	13
2.1.7 Li’s Method.....	14
2.1.8 Juang’s Method.....	14
2.1.9 Gutierrez’s Method	15
2.1.10 De Caro’s Method.....	15
2.1.11 Summary on Accuracy	17
2.2 Tables-and-additions Methods	20
2.2.1 Add-Table Lookup-Add (ATA) Method.....	21
2.2.2 Symmetric Bipartite Table Method (SBTM)	23
2.2.3 Multipartite Table Method (MTM)	24
2.2.4 Summary on Accuracy.....	25
CHAPTER 3. Unified Mitchell-based Approximation for Efficient Logarithmic Conversion..	27

3.1	Overview	27
3.2	Architectural Model and Design Parameters	28
3.2.1	Impact on Accuracy	30
3.2.2	Impact on Hardware Merits	31
3.3	Design Space Delimitation.....	33
3.4	Design Space Exploration	35
3.5	Division of Regions	36
3.6	Selection of POTTs.....	37
3.7	Error Analysis	39
3.8	Selection of Gradient	40
3.9	Algorithm.....	44
3.10	Performance Evaluation and Discussions	45
3.10.1	Comparisons with Mitchell-based Methods.....	45
3.10.2	Scalability of UMA.....	48
3.11	Summary	51
CHAPTER 4. Range Mapping – A Fresh Approach to High Accuracy Mitchell-based Logarithmic Conversion		53
4.1	Overview	53
4.2	Range Mapping (RM)	53
4.3	Piecewise Linear Approximation by Heuristic Approach.....	60
4.3.1	Accuracy	62
4.3.2	Hardware Complexity	64
4.3.3	Figures of Merit	66
4.4	Piecewise Linear Approximation by UMA.....	66
4.4.1	Modifications to UMA for Focus Range.....	67
4.4.2	Performances Evaluation	68
4.5	Discussions on Additional Mapping Option	72
4.5.1	Equivalence Property	72
4.5.2	Additional Mapping Option	73
4.6	Summary	74
CHAPTER 5. A Memory Efficient Tables-and-Additions Method for Accurate Computation of Elementary Functions		75
5.1	Overview	75

5.1.1	Approximation without CDF	75
5.1.2	Carry Prevention/Select Technique.....	76
5.1.3	Symmetry Property Exploitation (SPE)	76
5.1.4	Unequal partitioning of input	76
5.2	Mathematical Formulation.....	77
5.2.1	Approximation without CDF	77
5.2.2	Carry Prevention / Select Technique.....	80
5.2.3	Symmetry Property Exploitation (SPE)	82
5.2.4	Final Formulation with Unequal Partitioning of Input.....	85
5.3	Error Analysis	85
5.3.1	Approximation Error ϵ_a	86
5.3.2	Rounding Error ϵ_r	87
5.3.3	Derivations	88
5.4	Hardware Architecture.....	90
5.4.1	Tables.....	90
5.4.2	NOT and XOR Gates	91
5.5	Performance Evaluations for Binary Logarithm Function	92
5.5.1	Table Size.....	92
5.5.2	Standard Cell Implementation.....	94
5.6	Performance Evaluations for Other Elementary Functions.....	94
5.6.1	Table Size.....	94
5.6.2	FPGA Implementations.....	96
5.7	Summary	98
CHAPTER 6.	Discussions, Conclusions and Future Works	99
6.1	Discussions.....	99
6.2	Conclusions	99
6.3	Future Works.....	102
Author's Publications.....		104
Journal Publications		104
Conference Publications.....		104
Bibliography		105

Summary

Logarithmic Number System (LNS) is often used in digital signal processing to simplify complex arithmetic operations. LNS requires data to be converted into the logarithmic domain, i.e., logarithmic conversion. The thesis studies the VLSI architectures for logarithmic conversion.

The existing Mitchell-based logarithmic conversion methods are first investigated, specifically the parameters that impact the accuracy and hardware merits are analyzed and an architectural model is established. Subsequently, a novel method, named the Unified Mitchell-based Approximation method (UMA), is proposed to obtain, for any given accuracy requirement up to 14 bits, a logarithmic converter optimized in area-delay product (ADP). The experimental results show that the designs obtained have better ADPs than those by the existing Mitchell-based methods, with eight of the thirteen designs achieving better ADP by more than 50%. The designs with conversion accuracy of 12, 13 and 14 bits, which are not currently available, are also obtained.

Approaching the Mitchell-based methods from a fresh angle, a novel method named the Range Mapping (RM) is proposed to map and compress the entire range of approximation, $[1,2)$, of the logarithm function into only about one third of the original range. As a result, it enhances the effectiveness of the piecewise linear approximation for improving the accuracy. When a newly proposed 4-region piecewise linear approximation is developed for the compressed range, the maximum absolute error and maximum absolute error percentage are reduced by 15.0% and 25.0% respectively when compared to the best existing results. Furthermore, applying the UMA on the compressed range, optimized converters can be obtained for accuracy up to 18 bits.

For accuracy requirement higher than 18 bits, a new method known as the integrated Add-Table Lookup-Add (iATA) is proposed to reduce table (memory) size by abandoning the central difference formulation (used in the existing ATA method) for approximating the first derivatives in the Taylor series of the function. Three additional techniques, namely the carry select technique, the symmetry property exploitation and the unequal partitioning of input with the aid of error analysis, are integrated for further memory optimization. The experimental results show that iATA achieves a memory saving of 87.22% when compared to the best existing result.

The thesis details the development of the three methods, their mathematical formulations, the hardware architectures obtained, the performance evaluations of the architectures and their comparisons with the state-of-the-art designs.

List of Figures

Fig. 2-1: Generic hardware architecture for Mitchell-based methods	9
Fig. 2-2: Mitchell's straight line approximation of binary logarithm	9
Fig. 2-3: Juang's approximation of $\log_2(1 + m)$	15
Fig. 2-4: Error percentage of the existing Mitchell-based methods	19
Fig. 2-5: Partitioning of input and the inputs to tables for: (a) ATA, (b) SBTM, (c) STAM/MTM and (d) the proposed iATA	20
Fig. 3-1: Generic architectural model for approximation	29
Fig. 3-2: Design space tree of the proposed UMA	35
Fig. 3-3: Divisions of regions	37
Fig. 3-4: Approximation circuit for: (a) (3.18), (b) (3.17)	42
Fig. 3-5: Adder tree for approximation circuits in Fig. 3-4	42
Fig. 3-6: MUX for operand I_1 : (a) without optimization, (b) with optimization	43
Fig. 3-7: Pseudo code of proposed UMA	45
Fig. 3-8: <i>ADP</i> of the approximation functions	49
Fig. 4-1: A range mapping example	54
Fig. 4-2: Algorithm for finding focus ranges	55
Fig. 4-3: Destination segments d_i obtained from range mapping with G_i from Table 4-2: (a) $G_2 = 2^0$; (b) $G_2 = 2^0 + 2^{-5}$	58
Fig. 4-4: Focus range	59
Fig. 4-5: Gradient g of the $\log_2(1 + m_2)$ curve	59
Fig. 4-6: Error percentage of the existing Mitchell-based methods and the proposed method ...	63

Fig. 4-7: Architecture of the proposed RM method	64
Fig. 4-8: Divisions of regions	67
Fig. 4-9: Generic architectural model for approximation circuit	69
Fig. 4-10: <i>ADP</i> of the approximation functions	70
Fig. 4-11: Range mapping based on Equivalence Property	73
Fig. 5-1: iATA architecture	91
Fig. 6-1: Area-delay product for logarithmic converters obtained by the proposed methods for accuracy requirements ranging from 4 bits to 24 bits	100

List of Tables

Table 2-1: Mean-square error, MAE and coefficients for logarithm approximation	12
Table 2-2: Accuracy comparison for the existing Mitchell-based methods	17
Table 2-3: Elementary functions computed for various wordlength	25
Table 3-1: Possible Range for Design Parameters ($n = 32$)	27
Table 3-2: Area and delay in unit-gate model	31
Table 3-3: impact of design parameters on accuracy and complexly	33
Table 3-4: An example of C_r for approximation function (5.3)	38
Table 3-5: Possible Gradient Values ($P = 2, V = 2^{-3}$)	39
Table 3-6: An example of C_r	40
Table 3-7: area and delay for circuit in Fig. 3-4	43
Table 3-8: Accuracy and complexity comparisons among the designs of the proposed method and the existing Mitchell-based methods	46
Table 3-9: Designs obtained by the proposed search algorithm for accuracies achieved by the existing designs	47
Table 3-10: Estimated and synthesis results of the hardware merits	49
Table 3-11: Designs obtained by the proposed search algorithm for accuracies of 4 bits to 14 bits	51
Table 4-1: Focus Ranges Found for $I=4$	57
Table 4-2: Selected and Computed Focus Range	58
Table 4-3: Values for constants D_1, D_2, D_3 and D_4	61
Table 4-4: Accuracy comparisons among the proposed method and the existing Mitchell- based methods	62

Table 4-5: Synthesis Results in STM90nm CMOS Process	65
Table 4-6: Figures of Merit	66
Table 4-7: Estimated and synthesis results of the hardware merits	69
Table 4-8: Designs obtained by applying the UMA on focus range F for accuracies of 14 bits to 18 bits	71
Table 5-1: Implementation Details for $\log_2(X)$ Generated by iATA	92
Table 5-2: Table sizes for $\log_2(X)$ Generated by iATA	93
Table 5-3: Table size (bits) comparisons for $\log_2(X)$	
Table 5-4: Synthesis results for $\log_2(X)$	94
Table 5-5: Implementation Details for functions Generated by iATA with 24-bit input/output...	94
Table 5-6: Table sizes for functions Generated by iATA with 24-bit input/output	95
Table 5-7: Table size (bits) comparisons for $n = 24$	95
Table 5-8: Comparisons of Hardware implementation in Virtex II	96
Table 5-9: Implementation results in Virtex 7 with ISE 13.1	97

List of Abbreviations

ADP	Area-Delay product
ATA	Add-Table lookup-Add
CDF	Central Difference Formulation
CPT	Carry Prevention Technique
CST	Carry Select Technique
E	Error
EP	Error Percentage
EPR	Error Percentage Range
ER	Error Range
FA	Full Adder
FLP	Floating-Point
FoM	Figure of Merit
FXP	Fixed-Point
HA	Half Adder
iATA	integrated Add-Table lookup-Add
IC	Integrated Circuit
IO	Input/Output
LNS	Logarithmic Number System
LOD	Leading-One Detector
LSB	Least Significant Bit
LUT	Lookup table
MA	Mitchell-Based
MAE	Maximum Absolute Error
MAEP	Maximum Absolute Error Percentage
MI	Mitchell Interval
MILP	Mixed Integer Linear Programming
MNE	Maximum Negative Error
MNEP	Maximum Negative Error Percentage
MPE	Maximum Positive Error
MPEP	Maximum Positive Error Percentage
MSB	Most Significant Bit

MTM	Multipartite Table Method
MUX	Multiplexer
POTT	Power-Of-Two Term
RCA	Ripple Carry Adder
RM	Range Mapping
ROM	Read-Only Memory
SBTM	Symmetric Bipartite Table Method
SPE	Symmetry Property Exploitation
STAM	Symmetric Table Addition Method
ULP	Unit In The Last Place
UMA	Unified Mitchell-based Approximation
3D	Three-Dimensional

CHAPTER 1. Introduction

1.1 Background and Motivation

The evolution of integrated circuit (IC) technology especially technology scaling has become the main motivation for signal processing to be performed in digital domain via digital signal processing. Often, digital signal processing involves an enormous amount of complex arithmetic operations such as multiplication, division, square root and power. For example, 86.05% of the processing time of the rendering stage in a real-time three-dimensional (3D) graphic system is due to multiplication and division operations [Yos98]. Hardware realization of these complex operations usually consumes large silicon area, hungers for power and leads to long computation delays. This poses challenges for real-time, computationally intensive portable applications in which resources are limited in terms of processor performance, memory capacity and battery energy.

The Logarithmic number system (LNS) has been proposed as an alternative to fixed-point (FXP) and floating-point (FLP) number systems to simplify the complex arithmetic operations [Sto01]. For instance, multiplication and division operations can be simplified to addition and subtraction operations while square root and power operations can be simplified to shift and multiplication operations respectively. Consequently, the corresponding circuits are simplified. Moreover, LNS inherently possesses the simplicity characteristic of the FXP number system and the wide dynamic range characteristic of the FLP number system. However, the addition and subtraction in LNS are nonlinear functions. Research has been carried out to simplify the hardware realization of the logarithmic addition and subtraction operations [Arn90], [Arn07], [Mul98], [Swa75], [Swa83]. Alternatively, hybrid methods were proposed to perform complex operations in LNS and simple addition/subtraction operation in FXP/FLP number system [Hua08], [Lee05a], [Nam08], [Nam09].

LNS requires data to be converted into logarithmic domain, i.e., to compute the logarithm of the data (a.k.a. logarithmic conversion), using a logarithmic converter which incurs an additional hardware overhead. Theoretically, the logarithm of a data (other than the data that have value equal to the integer power of the base of the logarithm) has infinite precision. Nevertheless, in practice, the logarithm of a data is computed with finite precision/accuracy depending on the requirement of the applications because, in general, higher accuracy logarithmic conversion requires more complex converter and incurs higher hardware cost. Various hardware methods

have been proposed to implement logarithmic conversion. These methods can be broadly classified into three categories: 1) Mitchell-based, 2) lookup table (LUT)-based, and 3) digit recurrence.

To achieve both high speed and area efficient logarithmic conversion, Mitchell [Mit62] used a straight line to approximate every power-of-two interval of the logarithmic function. In each interval, the straight line deviates from the actual logarithm curve and hence resulting in relatively large conversion errors. Many methods based on the Mitchell method were subsequently proposed to improve the accuracy of approximation by trading off area, speed and/or power. They commonly divided the power-of-two intervals into 2, 3, 4, 6 or 8 regions and applied piecewise linear approximation in each region [Abe03], [Com65], [DeC11], [Gut11], [Hal70], [Jua09], [Kim06], [Li08], [San99], hence narrowing the deviation of the straight lines from the logarithm curve. The Mitchell-based methods are suitable for applications in which area, speed and power offset accuracy. Convergence analysis of Jacobi iterative method [Hua08], 3D graphics [Kim06], DCT/IDCT [Lee05a] for image compression and digital filter [Gup13] are some typical examples in which relaxation of accuracy requirement is exploited for power, area and delay saving. The logarithmic converters obtained by the existing Mitchell-based methods are having accuracies up to only less than 12 bits. In other words, it is inevitable that high accuracy methods are needed when high precision (more than 16 bits) logarithmic conversion is needed.

Digit recurrence methods, [DeC09], [Lan05], [Lo87], [Maj73], [Meh09], [Pin04], [Rod10], [Sum08], [Vol59], achieve high accuracy with efficient circuit area. However, due to their iterative nature, they are slow and/or have long latency. Therefore, they are not suitable for applications that require high computational efficiency. The LUT-based methods as the name imply, are memory intensive. They target for high accuracy applications. These LUT-based methods can also be used to compute various elementary functions such as reciprocal, exponential, natural logarithm and square root, which are crucial for applications namely direct digital frequency synthesizers, [Ash10], [Str07], Phong shaders, [Lai02], [Shi01], geometrical transformation, [Kar01a], [Kar01b], and N-body simulations [Liu00]. In general, the LUT-based methods are elementary function computation methods. They apply numerical methods such as polynomial interpolation and polynomial approximation to evaluate the elementary functions as follows:

$$\begin{aligned} f(X) &\approx a_0 + a_1X + a_2X^2 + \cdots + a_{k-1}X^{k-1} + a_kX^k \\ &= T_0 + T_1 + T_2 + \cdots + T_{k-1} + T_k \end{aligned} \tag{1.1}$$

where a_i , $i = \{0, 1, \dots, k\}$, are coefficients to be determined depending on the type of approximation/interpolation chosen. The common approximations used include Taylor series approximation, Chebyshev approximation and minimax polynomials [Phi96]. Lagrange interpolation and Central Difference Formulation are usually considered for interpolation [Phi96]. The LUT-based methods are further divided into two categories: the piecewise table lookup methods and the tables-and-additions methods. In the piecewise table lookup methods, [Ali10], [Cao01], [Det05] [Lam03], [Lee05b], [Lee07], [Lee08], [Lew94], [McC03], [Pal00], [Pin05], [Sas07], [Str11], [Tak98] [Wal05], the function is divided into several segments, each represented by a polynomial in the form of (1.1). Each term T_i is computed at run time before they are summed to produce the final results. On the other hand, the tables-and-additions methods, [Das95], [deD05], [Mul99], [Sch99], [Sti99], [Won95], need only lookup tables and adders. Each T_i is directly stored in a table addressed by the combination of different parts of the input and the outputs of the tables are then summed. Between the piecewise table lookup and the tables-and-additions methods, the former is more compact in size but slower in computation speed because multipliers are required. The latter is faster in computation speed but larger in size because large ROM-based lookup tables are needed to store the terms T_i . The speed advantage is evident from the study in [Hsi08] that the multipartite method [deD05] of the tables-and-additions category is 17% to 35% faster when compared to the piecewise table lookup method [Wal05] for functions such as reciprocal, square root, binary logarithm and exponential with wordlengths ranging from 8 bits to 24 bits. In general, the tables-and-additions methods are most suitable for speed critical applications. One instance is the use of the Multipartite Table Method [deD05] for fast conversion of Cartesian to Polar coordinates [Str08].

Given an accuracy requirement, it would be ideal if the logarithmic conversion could be implemented with minimum hardware overhead and maximum speed. This is important especially for applications such as 3D graphic processors (e.g. NVIDIA® Tesla® GPUs [NV11]) in which there are hundreds or even thousands of cores with each core containing more than one ALU that can be implemented in LNS using Mitchell-based conversion method as presented in [Kim06]. Even a small amount of saving in hardware merits such as area cost and/or critical path delay can easily result in significant performance improvement. As pointed out in [DeC11], the existing Mitchell-based methods do not employ a rigorous approach and, hence, could not ensure that the logarithmic converters are fully optimized for size and speed in hardware implementation. Besides, the logarithmic converters obtained by the existing Mitchell-based methods have accuracies below 12 bits and the methods are not scalable. A scalable method is a method that

could provide higher conversion accuracy for error-sensitive applications while keeping the hardware minimum and for error-insensitive applications, the conversion accuracy could be reduced easily so that no additional hardware would be incurred to achieve unnecessary high accuracy. The existing Mitchell-based methods are not scalable because the approach to find the approximation function is either not given, [Abe03], [Com65], [Gut11], [Kim06], or is meant to obtain only a specific approximation function, [Hal70], [Jua09], [Li08], [San99]. The only existing method that is scalable and uses a rigorous systematical approach is the mixed-integer linear programming based method reported in [DeC11], where the accuracy achieved is about 11 bits. It aims to minimize the maximum absolute error percentage (*MAEP*) for a given number of regions of approximation and hence the approximation function is not optimized for hardware implementation. On the other hand, for high accuracy and high speed logarithmic conversion methods specifically the tables-and-additions methods, the table size is impractically large. There is a limit in input wordlength and beyond which the required ROM becomes impractically large to be synthesized by the industrial tools [Hsi08]. In short, there is room for improvement in hardware merits of logarithmic converter obtained by the existing methods and therein lies the motivation behind this research work.

1.2 Research objectives

Broadly, the work is to explore methods and develop architectures that are efficient in both hardware complexity and computational speed for logarithmic conversion. The prime objective is to enhance the hardware merits of the logarithmic converters for accuracy requirements up to 24 bits. To achieve this main objective, the following issues have been identified and focused on in this research.

- a) To investigate and enhance the advantageous features of hardware simplicity and fast computation of the Mitchell-based methods, hence, to develop scalable methods for efficient logarithmic conversion to achieve accuracy higher than the best existing accuracy of 12 bits with optimized ADP. Specifically, this work aims to improve the ADP by at least 50% for 50% of the known existing Mitchell-based logarithmic converters for the same accuracy requirements.
- b) To extend the advantageous features of the Mitchell-based methods and develop new methods for achieving accuracy higher than the best accuracy of 12 bits by the existing Mitchell-based methods.

c) To investigate and reduce the table size of the tables-and-additions methods by at least 50% for efficient logarithmic conversion with accuracy higher than 16 bits, which cannot be achieved by the Mitchell-based methods.

1.3 Major contributions

The main contributions of the research work performed are summarized as follows.

a) The existing Mitchell-based methods for logarithm conversion are studied. A generic hardware model is formulated for the piecewise linear approximation function that is needed in the logarithmic conversion, and the parameters that impact the accuracy and hardware merits are analyzed. Subsequently, a novel method, named the Unified Mitchell-based Approximation method (UMA), is proposed to obtain, for any given accuracy requirement up to 14 bits, an approximation function that is optimized for hardware implementation. In the UMA, it is proposed to explore a design space consisting of five design parameters which impact the conversion accuracy as well as the area cost and critical path delay of the circuit for the approximation function. As the design space delimited by these five design parameters is huge, constraints are proposed on the design parameter values and a systematical search algorithm is introduced to obtain an optimized feasible approximation function practically. The experimental results show that the designs (logarithmic converters) obtained by the UMA for the same accuracy requirements achieved by the existing Mitchell-based designs have the same or better area-delay products (*ADPs*). Eight of the thirteen designs obtained achieve improvement in *ADP* by more than 50%. Besides, the designs with conversion accuracy of 12, 13 and 14 bits, which are not currently available, are obtained.

The proposed UMA is described in details in Chapter 3 as well as in the manuscript below.

[Low13a] **J. Y. L. Low**, and C. C. Jong, “Unified Mitchell-based Approximation for efficient logarithmic conversion circuit,” *IEEE Transactions on Computers*, under review.

b) A novel method named the Range Mapping (RM) method is proposed to map and compress the entire range of approximation, $[1,2)$, of the logarithm function into only about one third of the original range, hence reducing the overall curvature of the logarithm function by restricting the function to within the compressed range before applying the piecewise linear approximation. As a result, it enhances the effectiveness of the piecewise linear approximation in improving the accuracy in terms of error and error percentage. When a newly proposed 4-region piecewise linear approximation is developed by heuristic approach for the compressed range, the maximum absolute error and maximum absolute error percentage of the binary logarithm computation are

reduced by 15.0% and 25.0% respectively when compared to the best existing results obtained by Kim et al. [Kim06] and De Caro et al. [DeC11] respectively.

The proposed RM is presented in Chapter 4 and the manuscript below.

[Low13b] **J. Y. L. Low**, and C. C. Jong, “Range Mapping - A fresh approach to high accuracy Mitchell-based logarithmic conversion,” *IEEE Transactions on VLSI*, under review.

c) The proposed UMA and RM methods are combined in such a way that the systematical search algorithm proposed in the UMA method is applied on the compressed range determined by the RM method, to obtain, for any given accuracy requirement from 15 to 18 bits, an approximation function that is optimized for efficient hardware implementation. Note that the existing Mitchell-based methods only achieve accuracies less than 12 bits.

The application of the proposed UMA on the compressed range obtained by the RM method is discussed in details in Section 4.4.

d) A new method known as the integrated Add-Table Lookup-add (iATA) is proposed for logarithmic conversion as well as elementary function computation. To reduce total table size, the iATA method abandons the central difference formulation (CDF) that is employed by the existing method, Add-Table Lookup-Add (ATA) method [Won95] of the tables-and-additions category, to approximate the first derivatives in the Taylor series of the function to be evaluated. Furthermore, three techniques, namely the carry select technique (CST), the symmetry property exploitation (SPE) and the unequal partitioning of input with the aid of error analysis, are integrated in the iATA method for further memory optimization. The experimental results show that the proposed iATA achieves a memory saving of 87.22% for the computation of the logarithm function when compared to the best existing result obtained by SBTM [Sch99]. For the computation of the other elementary functions, 23.63% saving in memory is achieved for the reciprocal function when compared to the best existing result obtained by MTM [deD05]. For the natural logarithm function, iATA reduces the memory size by 61.39% when compared to the best existing result obtained by STAM [Sti99].

The proposed iATA is presented in Chapter 5 and in the following published paper.

[Low13c] **J. Y. L. Low**, and C. C. Jong, “A memory efficient tables-and-additions method for accurate computation of elementary functions,” *IEEE Transactions on Computers*, vol. 62, no.5, pp. 858-872, 2013.

1.4 Organization of the thesis

This thesis is organized into six chapters. In Chapter 1, the motivation, objectives and main contributions of the research work are detailed.

In Chapter 2, the related existing works, specifically the Mitchell-based logarithmic methods and the tables-and-additions-based of the LUT-based methods, are reviewed in details.

The main contributions of this research work are presented in Chapter 3 to 5. Chapter 3 describes the proposed Unified Mitchell Approximation (UMA) method which features a systematical search algorithm to obtain an optimized piecewise linear approximation that is needed in a logarithmic conversion for any accuracy requirement up to 14 bits within feasible period of time. In Chapter 4, a novel approach to the Mitchell-based methods, named the Range Mapping (RM) method, is presented. The Range Mapping method compresses the range of approximation of the piecewise linear approximation and thus enhances the effectiveness of the approximation. This chapter also describes the combination of the proposed UMA and RM methods to obtain an optimized piecewise linear approximation for any accuracy requirement up to 18 bits. Chapter 5 details the proposed integrated Add-Table lookup-Add (iATA) method which aims to achieve high accuracy logarithmic conversion with high speed yet memory efficient circuits.

Lastly, Chapter 6 summarizes the key results achieved by this research work and discusses the future works.

CHAPTER 2. Literature Review

The Mitchell-based methods and the tables-and-additions methods are two of the main categories for efficient logarithmic conversion. While the Mitchell-based methods have low complexity and high conversion speed but suffer from limited conversion accuracy, the tables-and-additions methods achieve fast and high accuracy conversion but require large memories. The two sections below review the existing works in these two categories.

2.1 Mitchell-based Methods

The Mitchell-based methods are high speed and area efficient but low accuracy logarithmic conversion methods. In this section, the Mitchell method is first reviewed followed by its descending methods each of which is covered in details in each subsection.

2.1.1 Mitchell's Straight Line Approximation

Mitchell's straight-line approximation of the logarithm of a binary number [Mit62] is summarized below.

Mathematically, an n -bit unsigned binary number N (2.1) in the interval of $0 \leq N \leq (2^n - 1)2^j$, where $j, k \in \mathbb{Z}, j \leq 0, n > 0 \in \mathbb{N}, j \leq k \leq n - 1 + j$ and $x_i \in \{0, 1\}$, can be expressed as

$$N = x_k \dots x_2 x_1 x_0 \cdot x_{-1} x_{-2} x_{-3} \dots x_j \quad (2.1)$$

$$= \sum_{i=j}^k 2^i x_i \quad (2.2)$$

Without losing generality, x_k can be considered as '1' because it is the leading one bit and (2.2) can be rewritten as

$$\begin{aligned} N &= 2^k + \sum_{i=j}^{k-1} 2^i x_i \\ &= 2^k (1 + \sum_{i=j}^{k-1} 2^{i-k} x_i) \\ &= 2^k (1 + m) \end{aligned} \quad (2.3)$$

where $m = \sum_{i=j}^{k-1} 2^{i-k} x_i$ is in the range of $[0, 1)$. Applying base-2 logarithm, (2.3) becomes

$$\log_2(N) = k + \log_2(1 + m) \quad (2.4)$$

Fig. 2-1: Generic hardware architecture for Mitchell-based methods

Fig. 2-2: Mitchell's straight line approximation of binary logarithm

where $k = \lfloor \log_2 N \rfloor$ and $\log_2(1 + m)$ are respectively known as the characteristic and mantissa of the base-2 logarithm of N . The value of k is the power of the leading-one position in N and can be obtained with a leading-one detector (LOD) and a small look-up table as illustrated in Fig. 2-1. $\log_2(N)$ can be obtained after $\log_2(1 + m)$ is computed. Mitchell approximated $\log_2(1 + m)$ with a straight line, $LA_{1.Mitchell}(1 + m)$, as follows.

$$\begin{aligned} \log_2(1 + m) &\cong LA_{1.Mitchell}(1 + m) \\ &= a \times m + b \\ &= m \end{aligned}$$

where gradient $a = 1$ and constant $b = 0$. The value m is produced by the logarithmic shifter and subsequently concatenated directly with k without passing through the approximation circuit as in Fig. 2-1. Therefore, Mitchell's straight-line approximation of logarithm is expressed as

$$\log_2(N) \cong k + LA_{1.Mitchell}(1 + m) \quad (2.5)$$

$$= k + m. \quad (2.6)$$

m is known as the Mitchell fraction in the literature. The Mitchell's straight-line approximation can be better visualized in Fig. 2-2 as the solid straight lines, each connecting two successive power-of-two points. In short, the Mitchell method divides the entire input range, $N = [0, 2^n)$,

into intervals, each called a Mitchell Interval (*MI*), between every two successive power-of-two points, 2^k and 2^{k+1} for $k = 0, 1, \dots, n-1$, denoted as $MI_k = [2^k, 2^{k+1})$. Essentially, as shown in (2.4), the problem of approximating $\log_2(N)$ is now reformulated by Mitchell as the problem of approximating $\log_2(1 + m)$ and the range of approximation is always $1 \leq 1 + m < 2$, referred to as the Mitchell Range and denoted as $M = [1, 2)$, regardless of the *MI*s.

The Mitchell method is efficient for implementation in integrated circuit but suffers from a maximum absolute error percentage (2.10) of more than 5%. Many Mitchell-based methods [Abe03], [Com65], [DeC11], [Gut11], [Hal70], [Jua09], [Kim06], [Li08], [San99] have been proposed to improve the accuracy of the Mitchell's straight line approximation. Generally, these Mitchell's descending methods divide the Mitchell range $M = [1, 2)$ into certain number of regions and have individual linear approximation function for each region implemented as the approximation circuit in the dotted-edge box of Fig. 2-1. They distinguish among themselves in the number of regions and the piecewise linear approximation functions (a.k.a. error correction equations) in the regions that are used to approximate $\log_2(1 + m)$. In general, the approximation functions are expressed as (2.7) and/or (2.8).

$$\begin{aligned} \log_2(1 + m) &\cong LA_{R,\#}(1 + m) \\ &= a_r \times m + b_r \end{aligned} \quad (2.7)$$

$$= \begin{cases} m + C_r \times m_{MSBh} + D_r, & a_r \geq 1 \\ m + C_r \times \bar{m}_{MSBh} + D_r, & a_r < 1 \end{cases} \quad (2.8)$$

where m_{MSBh} is the h number of most significant bits (MSBs) of m , $\bar{m}_{MSBh} = 1 - m_{MSBh} - 2^{-h}$ is the 1's complement of m_{MSBh} , and a_r , b_r , C_r and D_r are the coefficients of the approximation functions in the r -th region, g_r , $r \in \{1, 2, \dots, R\}$ and R is the number of regions of the approximation. Note that the subscription '#' in $LA_{R,\#}(1 + m)$ is the name of the first author (e.g. Mitchell in (2.5)) or a unique letter for distinguishing different approximation functions. In different Mitchell-based methods, different values are proposed for the coefficients in each region. The major Mitchell-based methods are reviewed in the following subsections. In these Mitchell-based methods, the maximum absolute error, *MAE*, and the maximum absolute percentage error, *MAEP*, are defined in (2.9) and (2.10) respectively.

$$MAE = \max_{\forall m} \{ |\log_2(1 + m) - LA_{R,\#}(1 + m)| \} \quad (2.9)$$

$$MAEP = \max_{\forall N} \left\{ \frac{|\log_2(1 + m) - LA_{R,\#}(1 + m)|}{\log_2(N)} \times 100\% \right\} \quad (2.10)$$

2.1.2 Combet's Method

Combet et al. in [Com65] divided the Mitchell range, $M = [1, 2)$, into four regions and proposed a 4-region approximation as follows.

$$LA_{4, \text{Combet}} = \begin{cases} m + (2^{-2} + 2^{-4})m & m \in [0.0, 0.25) \\ m + \frac{5}{64} & m \in [0.25, 0.5) \\ m + 2^{-3}\bar{m} + \frac{3}{128} & m \in [0.5, 0.75) \\ m + 2^{-2}\bar{m} & m \in [0.75, 1.0) \end{cases} \quad (2.11)$$

Starting with the general expression (2.12), which is similar to the form of (2.8),

$$m + C_r \times f(m) + D_r \quad (2.12)$$

(2.11) was determined by trial and error to make the deviations of straight lines (2.12) from $\log_2(1 + m)$ small subjecting to two considerations as follow: 1) If the slope of $\log_2(1 + m)$ for a region is greater than 1, then $f(m) = m$ is used. Otherwise, if the slope is less than 1, $f(m) = 1 - m$ is applied. However, for simple hardware implementation in circuit, $f(m) = 1 - m$ is replaced with $\bar{m}$ and hence $f(m) = 1 - m - 2^{-(LSB)}$, and 2) The coefficients C_r and D_r are chosen to be fractions with integer for numerator and a power-of-two value for denominator. This also simplifies the hardware implementation as both integer numerator and power-of-two denominator can be easily implemented by simple shifters.

It is reported in [Com65] that the *MAE* and *MAEP* are 0.008 and 0.267% respectively.

2.1.3 Hall's Method

Hall et al. in [Hal70] partitioned the Mitchell range M into 1 (no partitioning), 2, 4 and 8 regions. In Hall et al.'s method, (2.7) is first used to approximate the value of $\log_2(1 + m)$. Unlike Combet et al.'s method in which the coefficients are found by trial and error, the coefficients a_r and b_r in Hall et al.'s method are determined such that (2.7) will have linear least squares fit to $\log_2(1 + m)$ in each region as follows. The mean-square error, $\bar{e}_3$, is defined as

$$\bar{e}_3 = \frac{1}{m_2 - m_1} \int_{m_1}^{m_2} \{\log_2(1 + m) - (a_r \times m + b_r)\}^2 dm$$

where $0 \leq m_1 \leq m \leq m_2 < 1$. By making the first derivatives of $\bar{e}_3$, $\frac{\partial \bar{e}_3}{\partial a_r}$ and $\frac{\partial \bar{e}_3}{\partial b_r}$, equal to zero, $\bar{e}_3$ can be minimized with respect to a_r and b_r . By solving the first derivatives, the coefficients a_r and b_r were determined as tabulated in Table 2-1, which also includes the mean-square error $\bar{e}_3$ and *MAE* for each region of approximation.

Table 2-1: Mean-square error, MAE and coefficients for logarithm approximation

No. of regions	regions	a_r	b_r	$\bar{e}_3$	MAE
1	1	0.9843	0.0652	$0.6411 \cdot 10^{-3}$	0.06518
2	1	1.1636	0.0213	$0.1929 \cdot 10^{-4}$	0.02130
	2	0.8278	0.1816	$0.5817 \cdot 10^{-5}$	0.01052
4	1	1.2856	0.0062	$0.2782 \cdot 10^{-6}$	0.00624
	2	1.0510	0.0633	$0.3871 \cdot 10^{-6}$	0.00414
	3	0.8888	0.1435	$0.6425 \cdot 10^{-6}$	0.00219
	4	0.7702	0.2319	$0.2899 \cdot 10^{-7}$	0.00219
8	1	1.3592	0.0017	$0.1733 \cdot 10^{-7}$	0.00168
	2	1.2154	0.0194	$0.5509 \cdot 10^{-7}$	0.00137
	3	1.0994	0.0482	$0.8142 \cdot 10^{-7}$	0.00113
	4	1.0039	0.0839	$0.2972 \cdot 10^{-7}$	0.00093
	5	0.9234	0.1240	$0.1301 \cdot 10^{-6}$	0.00079
	6	0.8547	0.1669	$0.1287 \cdot 10^{-6}$	0.00070
	7	0.8070	0.2020	$0.1868 \cdot 10^{-6}$	0.00123
	8	0.7341	0.2658	$0.1500 \cdot 10^{-6}$	0.00119

The values of the coefficients a_r and b_r in Table 2-1 have infinite precision and need to be truncated for practical hardware implementation. In [Hal70], Hall et al. only used 4-region approximation as an example and truncated the coefficients at bit position 2^{-7} as shown in (2.13). For (2.13), only MAE of 0.0094 is reported in [Hal70].

$$LA_{4.Hall} = \begin{cases} m + (2^{-2} + 2^{-5} + 2^{-7})m + \frac{1}{128} & m \in [0.0, 0.25) \\ m + (2^{-5} + 2^{-6})m + \frac{1}{16} & m \in [0.25, 0.5) \\ m + (2^{-4} + 2^{-5} + 2^{-6})\bar{m} + \frac{1}{32} & m \in [0.5, 0.75) \\ m + (2^{-3} + 2^{-4} + 2^{-5} + 2^{-7})\bar{m} & m \in [0.75, 1.0) \end{cases} \quad (2.13)$$

2.1.4 SanGregory's Method

In [San99], SanGregory et al. proposed to divide the Mitchell range M into two regions as follows.

$$LA_{2.SanGregory} = \begin{cases} m + 2^{-2}m_{MSB4} & m \in [0, 0.5) \\ m + 2^{-2}\bar{m}_{MSB4} & m \in [0.5, 1) \end{cases} \quad (2.14)$$

SanGregory et al.'s method is based on the fact that the $MAEP$ for the Mitchell method [Mit62] occurs at the midpoint of M , $m = 0.5$. Observation reveals that, at this midpoint, the bit pattern of the mantissa, $\log_2(1 + m)$, is "10010101" while the bit pattern of the Mitchell fraction, m , is "1000000". Comparing two bit patterns, SanGregory reported that the difference in the fourth bit alone accounts for 73.6% of the MAE . Also it is important to note that the MSB of m is "1".

Therefore, when $m = 0.5$ is approached in the region $m = [0,0.5)$ and if m is shifted three bits to the right before adding back to the Mitchell fraction, the fourth bit of the new Mitchell fraction will approach “1”. For the region $m = [0.5,1)$, similar technique is used except that the shifted bits are inverted before the addition. Subsequently, SanGregory et al. performed VHDL simulations and revealed that the optimized approximation is achieved by shifting only the first four bits of m to the right by two bits before addition resulting in approximation function in (2.14).

According to SanGregory et al. in [San99], a *MAEP* of 1.54% is achieved by the 2-region approximation (2.14).

2.1.5 Abed’s Method

In [Abe03], Abed and Siferd proposed 2, 3 and 6-region approximations as follows.

$$LA_{2.Abed} = \begin{cases} m + 2^{-2}m_{MSB3} & m \in [0,0.5) \\ m + 2^{-2}\bar{m}_{MSB3} & m \in [0.5,1) \end{cases} \quad (2.15)$$

$$LA_{3.Abed} = \begin{cases} m + 2^{-2}m_{MSB4} & m \in [0.00,0.25) \\ m + \frac{5}{64} & m \in [0.25,0.75) \\ m + 2^{-2}\bar{m}_{MSB4} & m \in [0.75,1.00) \end{cases} \quad (2.16)$$

$$LA_{6.Abed} = \begin{cases} m + 2^{-2}m_{MSB6} & m \in [0.00,0.0625) \\ m + 2^{-2}m_{MSB6} + \frac{1}{64} & m \in [0.0625,0.25) \\ m + \frac{19}{256} & m \in [0.250,0.375) \\ m + \frac{11}{128} & m \in [0.375,0.625) \\ m + \frac{9}{128} & m \in [0.625,0.750) \\ m + 2^{-2}\bar{m}_{MSB6} & m \in [0.750,1.000) \end{cases} \quad (2.17)$$

As stated in [Abe03], the values of the coefficients as well as the number of MSBs of the Mitchell fraction, m , used for in the approximations are determined via software simulations. Small error percentage, low area cost and high speed conversion are the main optimization factors in the simulations.

The *MAEP* for 2, 3 and 6-region approximations are reported in [Abe03] to be 0.9299%, 0.4314% and 0.1538% respectively.

2.1.6 Kim’s Method

A 8-region approximation was proposed by Kim et al. in [Kim06] as follows.

$$\begin{aligned}
& \text{LA}_{8,\text{Kim}} = \\
& \begin{cases} m + 2^{-1}m_{\text{MSB}24} + 2^{-3}\bar{m}_{\text{MSB}22} + 2^{-7}\bar{m}_{\text{MSB}18} & m \in [0.00, 0.125) \\ m + 2^{-1}m_{\text{MSB}24} + 2^{-2}\bar{m}_{\text{MSB}23} + 2^{-6}\bar{m}_{\text{MSB}19} + \frac{20}{1024} & m \in [0.125, 0.25) \\ m + 2^{-2}m_{\text{MSB}23} + 2^{-3}\bar{m}_{\text{MSB}22} + 2^{-6}\bar{m}_{\text{MSB}19} + \frac{46}{1024} & m \in [0.25, 0.375) \\ m + 2^{-7}\bar{m}_{\text{MSB}18} + \frac{84}{1024} & m \in [0.375, 0.50) \\ m + 2^{-4}m_{\text{MSB}21} + 2^{-3}\bar{m}_{\text{MSB}22} + 2^{-7}\bar{m}_{\text{MSB}18} + \frac{123}{1024} & m \in [0.50, 0.625) \\ m + 2^{-3}\bar{m}_{\text{MSB}22} + 2^{-6}\bar{m}_{\text{MSB}19} + \frac{167}{1024} & m \in [0.625, 0.75) \\ m + 2^{-4}m_{\text{MSB}21} + 2^{-2}\bar{m}_{\text{MSB}23} + 2^{-6}\bar{m}_{\text{MSB}19} + \frac{215}{1024} & m \in [0.75, 0.875) \\ m + 2^{-2}\bar{m}_{\text{MSB}23} + 2^{-7}\bar{m}_{\text{MSB}18} + \frac{264}{1024} & m \in [0.875, 1.00) \end{cases} \quad (2.18)
\end{aligned}$$

In [Kim06], the $\text{MAEP} = 0.05\%$ of the approximation function (2.18) is reported but no detail is provided on the approach to obtain the approximation.

2.1.7 Li's Method

Similar to the Mitchell method, Li et al. [Li08] used only one straight line (2.19) to approximate the value of $\log_2(1 + m)$ for the Mitchell range M .

$$\text{LA}_{1,\text{Li}} = m + 0.0430355 \quad (2.19)$$

Comparing (2.19) and (2.7), it can be seen that $a_1 = 1$ and $b_1 = 0.0430355$. Recall that, in the Mitchell method (Section 2.1.1), b is equal to zero and the MAE is 0.086071 as reported in [Li08] (0.08639 in [Abe03]). Also, it is noted that the error due to the Mitchell method is positive for the Mitchell range M . By observing the behavior of the error, Li et al. proposed $b = 0.0430355$ to reduce the MAE to about half of the MAE of the Mitchell method. In geometrical interpretation, Li's method can be illustrated as a vertical shift of the Mitchell's straight line of approximation.

2.1.8 Juang's Method

Juang et al. discovered that the quadrangle $abcp$ in Fig. 2-3 is a parallelogram, where the straight lines $\overline{ab}$ and $\overline{bc}$ have the same slope as that of $\overline{pc}$ and $\overline{ap}$ respectively. The two sides, $\overline{ap}$ and $\overline{pc}$, of the parallelogram trace the $\log_2(1 + m)$ tightly and touch the curve at points a , p and c . Furthermore, the slope of $\overline{ab}$ and $\overline{pc}$ is the reciprocal of the slope of $\overline{ap}$ and $\overline{bc}$. Capitalizing on these properties of the curve, Juang et al. proposed a 2-region approximation as follows.

$$\text{LA}_{2,\text{Juang}} = \begin{cases} m + (2^{-3} + 2^{-4})m_{\text{MSB}3} & m \in [0, 0.5) \\ m - (2^{-3} + 2^{-5})m_{\text{MSB}3} + \frac{5}{32} & m \in [0.5, 1) \end{cases} \quad (2.20)$$

Fig. 2-3: Juang's approximation of $\log_2(1+m)$

Note that the slope in the region $[0,0.5)$ is $19/16$ ($1 + 2^{-3} + 2^{-4}$) while the slope in the region $[0.5,1)$ is $27/32$ ($1 - 2^{-3} - 2^{-5}$). $27/32 = 0.11011000_2$ is approximately equal to $16/19 = 0.11010111_2$, which is the reciprocal of the slope in the region $[0,0.5)$.

It is reported in [Jua09] that the *MAE* and the *MAEP* are 0.036 and 2.889% respectively. It is important to note that the percentage error is computed with N in the interval of $[2, 4)$ and the inputs are assumed to have 6 integer bits and 26 fraction bits.

2.1.9 Gutierrez's Method

Gutierrez and Valls proposed a 4-region approximation in [Gut11] as follows.

$$LA_{4,\text{Gutierrez}} = \begin{cases} m + 2^{-2}m_{MSB8} + 0.008 & m \in [0.0,0.25) \\ m + 2^{-5}m_{MSB8} + 0.07 & m \in [0.25,0.5) \\ m - 2^{-3}m_{MSB8} + 0.15 & m \in [0.5,0.75) \\ m - 2^{-2}m_{MSB8} + 0.25 & m \in [0.75,1.0) \end{cases} \quad (2.21)$$

where the constants are truncated at 2^{-13} binary position, i.e., the bits after 2^{-13} are discarded. The number of bits truncated in m and the coefficients of the approximation function (E.2.13.1) are determined by trial and error as stated in [Gut11] which provides the *MAE* of 0.78125 for (E.2.13.1).

2.1.10 De Caro's Method

In [DeC11], De Caro et al. proposed a rigorous approach to determine the coefficients, C_r and D_r in (2.8), of the piecewise linear approximation functions. Given the number of regions, R , De Caro

et al. applied mixed-integer linear programming (MILP) to find the optimized coefficient values that minimize the *MAEP* (2.10). Based on MILP, De Caro et al. proposed 2, 4 and 8-region approximations as follows.

$$\text{LA}_{2,\text{DeCaro}} = \begin{cases} m + (2^{-2} - 2^{-7})m_{MSB5} & m \in [0,0.5) \\ m + (2^{-4} - 2^{-2})m_{MSB5} + \frac{93}{512} & m \in [0.5,1) \end{cases} \quad (2.22)$$

$$\text{LA}_{4,\text{DeCaro}} = \begin{cases} m + (2^{-2} + 2^{-4})m_{MSB5} & m \in [0.0,0.25) \\ m + (2^{-4} + 2^{-6})m_{MSB5} + \frac{27}{512} & m \in [0.25,0.5) \\ m + (2^{-5} - 2^{-3})m_{MSB5} + \frac{68}{512} & m \in [0.5,0.75) \\ m - (2^{-2} + 2^{-5})m_{MSB5} + \frac{139}{512} & m \in [0.75,1.0) \end{cases} \quad (2.23)$$

$$\text{LA}_{8,\text{DeCaro}} = \begin{cases} m + (2^{-2} + 2^{-3} + 2^{-7})m_{MSB8} & m \in [0.00,0.125) \\ m + (2^{-2} - 2^{-6})m_{MSB8} + \frac{16}{1024} & m \in [0.125,0.25) \\ m + (2^{-3} - 2^{-6})m_{MSB8} + \frac{46}{1024} & m \in [0.25,0.375) \\ m + 2^{-7}m_{MSB8} + \frac{84}{1024} & m \in [0.375,0.50) \\ m - (2^{-4} + 2^{-7})m_{MSB8} + \frac{123}{1024} & m \in [0.50,0.625) \\ m - (2^{-3} + 2^{-7})m_{MSB8} + \frac{162}{1024} & m \in [0.625,0.75) \\ m + (2^{-4} - 2^{-2} - 2^{-6})m_{MSB8} + \frac{215}{1024} & m \in [0.75,0.875) \\ m + (2^{-7} - 2^{-2} - 2^{-5})m_{MSB8} + \frac{278}{1024} & m \in [0.875,1.00) \end{cases} \quad (2.24)$$

As reported in [DeC11], the approximation functions (2.22), (2.23) and (2.24) achieve the *MAEP* of 0.582%, 0.189% and 0.0324% respectively.

2.1.11 Summary on Accuracy

Table 2-2 summarizes the accuracy comparison for the all the Mitchell-based binary logarithmic conversion methods discussed in Sections 2.1.1 – 2.1.10. For each method, the number of regions, maximum positive error (*MPE*), maximum negative error (*MNE*), error range (*ER*), maximum positive error percentage (*MPEP*), maximum negative error percentage (*MNEP*) and error percentage range (*EPR*) are tabulated. The data in Table 2-2 are obtained from the literatures [Abe03], [Com65], [DeC11], [Gut11], [Hal70], [Jua09], [Kim06], [Li08], [Mit62], [San99] and/or through simulations in this research work. In the simulations, 32-bit integer input N is considered and the double-precision logarithm function in Matlab is used to emulate the accurate values of $\log_2(N)$. Fig. 2-4 depicts the simulation results obtained in this work for the error percentage of the logarithmic converters proposed in the existing Mitchell-based methods.

It is worth noting that there are discrepancies between the simulation results obtained in this research work and some of those reported in the literatures. For the 4-region approximation by Hall et al. [Hal70], this research work obtained the *MNEP* of -0.7812% (Table 2-2) while Abed and Siferd reported -0.7182% in [Abe03]. It can be seen from Fig. 2-4(c) (and Fig. 10(d) of [Abe03]) that the *MNEP* should be less than -0.75% . Abed and Siferd [Abe03] reported the *MPEP* and *MNEP* of 0.1538% and -0.1529% respectively for their 6-region approximation while this research work obtained -0.1538% and 0.1529% , i.e. sign discrepancy. The error percentage graph in Fig. 2-4(g) for Abed's 6-region approximation is inverted as compared to that of Fig. 10(g) in [Abe03]. Similar discrepancies occur in the maximum error percentages, *MPEP* and *MNEP*, of the 8-region approximation by Kim et al. where the graph in Fig. 2-4(h) is an inverted version of the graph in Fig. 7 of [Kim06]. The *MPE* and *MNE* for the 4-region approximation by Hall et al. were reported in [Hal70] as 0.0099 and -0.0078 respectively but this research work

TABLE 2-2: ACCURACY COMPARISON FOR THE EXISTING MITCHELL-BASED METHODS

Methods	Mitchell [Mit62] 1962	Combet [Com65] 1965	Hall [Hal70] 1970	San- Gregory [San99] 1999	Abed [Abe03] 2003			Kim [Kim06] 2006	Li [Li08] 2008	Juang [Jua09] 2009	Guti- errez [Gut11] 2011	De Caro [DeC11] 2011		
Regions	1	4	4	2	2	3	6	8	1	2	4	2	4	8
MPE ($\times 10^{-3}$)	86.1	8.0	4.4	29.3 [#]	44.9 [#]	29.3 [#]	13.2 [#]	2.0 ^{#*1}	43.0	37.0 [#]	7.2	22.9 [#]	16.0 [#]	2.8 [#]
MNE ($\times 10^{-3}$)	0	-6.2	-8.2	-28.0 [#]	-18.3 [#]	-20.8 [#]	-13.0 [#]	-1.5 [#]	-43.0	-9.8 [#]	-7.9	-28.6 [#]	-8.9 [#]	-2.5 [#]
ER ($\times 10^{-3}$)	86.1	14.2	12.6	57.3 [#]	63.2 [#]	50.1 [#]	26.2 [#]	3.5 [#]	86.0	46.8 [#]	15.1	51.5 [#]	24.9 [#]	5.3 [#]
MPEP (%)	5.361	0.185	0.126	0.431	0.930	0.431	0.153	0.050	2.645 [#]	1.079	0.181 [#]	0.582	0.189	0.032 ^{*2}
MNEP (%)	0	-0.267	-0.781	-1.540	-0.554	-0.268	-0.154	-0.044	-4.304 [#]	-0.096	-0.793 [#]	-0.498	-0.114	-0.031
EPR (%)	5.361	0.452	0.907	1.971	1.484	0.699	0.307	0.094	6.949 [#]	1.175	0.974 [#]	1.080	0.303	0.063

#: results not reported in any previous works; *1: best existing MAE ($=\max(|MPE|, |MNE|)$); *2: best existing MAEP ($=\max(|MPEP|, |MNEP|)$)

obtained 0.0044 and -0.0082 from simulations.

Among the existing Mitchell-based methods, the methods by Kim et al. and De Caro et al. achieve, respectively, the best results in the maximum absolute error ($MAE = \max(|MPE|, |MNE|)$) of 0.002 and maximum absolute error percentage ($MAEP = \max(|MPEP|, |MNEP|)$) of 0.032% with their 8-region approximations.

Fig. 2-4: Error percentage of the existing Mitchell-based methods

2.2 Tables-and-additions Methods

The tables-and-additions methods of the LUT-based category are used to compute not only the logarithm function but also various elementary functions with different range of approximation for high accuracy requirement. For the logarithm function, the range of approximation of $[1,2)$ is considered. In other words, they can be employed to compute $\log_2(1 + m)$, where $1 + m \in M = [1,2)$, which is required in the computation of $\log_2(N)$ according to (2.4). In general, the logarithm function is treated as one of the elementary functions in the tables-and-additions methods. As such, the methods are discussed and described as elementary function computation methods in this section and Chapter 5.

Under this tables-and-additions category, there are three main types of methods namely the Add-Table Lookup-Add (ATA) method, the Symmetry Bipartite Table Method (SBTM) and the Multipartite Table Method (MTM). The main characteristic of these methods is that no multiplier is needed in the implementations. Therefore, they are high speed elementary function computation methods. Although they share one common feature of storing T_i of (1.1) directly in tables and summing the table outputs to produce the final result, they have five major differences, explicitly the partitions of the input, the number of tables, the input to the tables, the entries of the tables and the table sizes. The first three differences are illustrated in Fig. 2-5, where the 24-bit input X is divided into several parts, X_i of length n_i bits for $i = \{0, 1, 2, \dots\}$. The newly proposed iATA

Fig. 2-5: Partitioning of input and the inputs to tables for: (a) ATA, (b) SBTM, (c) STAM/MTM and (d) the proposed iATA

method employs different input partition and has a new mathematical formulation for table size optimization as presented in Chapter 5. The table input $\{X_i, X_j\}$ denotes the concatenation of X_i and X_j while $\langle X_i, X_j \rangle$ denotes that some bits of X_i and X_j are used for the input as detailed in Section 5.4. Note that the wordlength of the table input also dictates the table size. The following sections summarize the three methods.

2.2.1 Add-Table Lookup-Add (ATA) Method

As the name suggested, the ATA method proposed by Wong and Goto [Won95] starts the computation with 4 parallel 2-operand additions/subtractions followed by 6 parallel table look-ups and ends with a 6-operand addition as illustrated in Fig. 2-5(a), where the input X is divided into four equal parts such that

$$X = X_0 + \lambda X_1 + \lambda^2 X_2 + \lambda^3 X_3.$$

i.e. $n_0 = 6$, $\lambda = 2^{-6}$ and $X_i = [0, 1 - 2^{-6}]$ for $i = \{0, 1, 2, 3\}$. The entries for the tables are derived from the mathematical formulation below, which also determines the inputs to the tables.

The Taylor series [Phi96] of a function $f(X)$ can be expressed as follows:

$$f(X) = f(X_0 + \lambda X_1) + f^{(1)}(X_0 + \lambda X_1)(\lambda^2 X_2 + \lambda^3 X_3) + \frac{f^{(2)}(X_0 + \lambda X_1)}{2!}(\lambda^4 X_2^2 + 2\lambda^5 X_2 X_3 + \lambda^6 X_3^2) + \dots \quad (2.25)$$

Ignoring the second and higher order derivatives, (2.25) can be approximated by $\tilde{f}(X)_{ATA}$ given in (2.26) as explained below. The first derivative $f^{(1)}(X_0 + \lambda X_1)(\lambda^2 X_2)$ in (2.25) is approximated by D_1^{ATA} in (2.27) and can be achieved by (2.28), which is obtained from the central difference formulation $f^{(1)}(A) \approx \frac{f(A+B) - f(A-B)}{2B}$ [Phi96] with $A = X_0 + \lambda X_1$ and $B = \lambda X_2$. Taylor series is then applied to (2.28) to obtain (2.29). It can be seen that the first term in (2.30) is the desired term $f^{(1)}(X_0 + \lambda X_1)(\lambda^2 X_2)$ and the remaining terms are the error terms due to the approximation. Similarly, the other first derivative $f^{(1)}(X_0 + \lambda X_1)(\lambda^3 X_3)$ in (2.25) is approximated by (2.31) and (2.33) is derived after applying the central difference formulation and Taylor series.

$$\tilde{f}(X)_{ATA} = D_0^{ATA} + D_1^{ATA} + D_2^{ATA} \quad (2.26)$$

where

$$\begin{aligned} D_0^{ATA} &= f(X_0 + \lambda X_1) \\ D_1^{ATA} &\approx f^{(1)}(X_0 + \lambda X_1)(\lambda^2 X_2) \end{aligned} \quad (2.27)$$

$$D_1^{ATA} = \frac{\lambda}{2} f(X_0 + \lambda X_1 + \lambda X_2) - \frac{\lambda}{2} f(X_0 + \lambda X_1 - \lambda X_2) \quad (2.28)$$

$$\begin{aligned} &= \frac{\lambda}{2} \{ [f(X_0 + \lambda X_1) + f^{(1)}(X_0 + \lambda X_1)(\lambda X_2) \\ &+ \frac{f^{(2)}(X_0 + \lambda X_1)}{2!} (\lambda^2 X_2^2) + \frac{f^{(3)}(X_0 + \lambda X_1)}{3!} (\lambda^3 X_2^3) + \dots] \\ &- [f(X_0 + \lambda X_1) - f^{(1)}(X_0 + \lambda X_1)(\lambda X_2) \\ &+ \frac{f^{(2)}(X_0 + \lambda X_1)}{2!} (\lambda^2 X_2^2) - \frac{f^{(3)}(X_0 + \lambda X_1)}{3!} (\lambda^3 X_2^3) + \dots] \} \end{aligned} \quad (2.29)$$

$$= f^{(1)}(X_0 + \lambda X_1)(\lambda^2 X_2) + \frac{f^{(3)}(X_0 + \lambda X_1)}{6} (\lambda^4 X_2^3) + \dots \quad (2.30)$$

$$D_2^{ATA} \approx f^{(1)}(X_0 + \lambda X_1)(\lambda^3 X_3) \quad (2.31)$$

$$D_2^{ATA} = \frac{\lambda^2}{2} f(X_0 + \lambda X_1 + \lambda X_3) - \frac{\lambda^2}{2} f(X_0 + \lambda X_1 - \lambda X_3) \quad (2.32)$$

$$= f^{(1)}(X_0 + \lambda X_1)(\lambda^3 X_3) + \frac{f^{(3)}(X_0 + \lambda X_1)}{6} (\lambda^5 X_3^3) + \dots \quad (2.33)$$

The error of approximating $f(X)$ by $\tilde{f}(X)_{ATA}$ is obtained by subtracting (2.26) from (2.25).

$$\begin{aligned} f(X) - \tilde{f}(X)_{ATA} &= \frac{f^{(2)}(X_0 + \lambda X_1)}{2} (\lambda^4 X_2^2 + 2\lambda^5 X_2 X_3 + \lambda^6 X_3^2) \\ &\quad - \frac{f^{(3)}(X_0 + \lambda X_1)}{6} (\lambda^4 X_2^3 + \lambda^5 X_3^3) + \dots \end{aligned} \quad (2.34)$$

In ATA, the terms with values smaller than λ^5 in (2.34) are ignored and two correction terms $\frac{f^{(2)}(X_0 + \lambda X_1)}{2} (\lambda^4 X_2^2) - \frac{f^{(3)}(X_0 + \lambda X_1)}{6} (\lambda^4 X_2^3)$ are added to (2.26) so that the approximation is accurate to λ^5 . To reduce the table size, the correction terms $f^{(2)}(X_0 + \lambda X_1)$ and $f^{(3)}(X_0 + \lambda X_1)$ are replaced with $f^{(2)}(X_0)$ and $f^{(3)}(X_0)$ respectively. As a result, the approximation equation for the ATA method is:

$$\begin{aligned} P_{ATA}(X) &= \tilde{f}(X)_{ATA} + \lambda^4 \left[\frac{f^{(2)}(X_0)}{2} (X_2^2) - \frac{f^{(3)}(X_0)}{6} (X_2^3) \right] \\ &= D_0^{ATA} + D_1^{ATA} + D_2^{ATA} + \lambda^4 \left[\frac{f^{(2)}(X_0)}{2} (X_2^2) - \frac{f^{(3)}(X_0)}{6} (X_2^3) \right] \\ &= f(X_0 + \lambda X_1) + \\ &\quad \frac{\lambda}{2} f(X_0 + \lambda X_1 + \lambda X_2) - \frac{\lambda}{2} f(X_0 + \lambda X_1 - \lambda X_2) + \\ &\quad \frac{\lambda^2}{2} f(X_0 + \lambda X_1 + \lambda X_3) - \frac{\lambda^2}{2} f(X_0 + \lambda X_1 - \lambda X_3) + \\ &\quad \lambda^4 \left[\frac{f^{(2)}(X_0)}{2} (X_2^2) - \frac{f^{(3)}(X_0)}{6} (X_2^3) \right] \end{aligned}$$

$P_{ATA}(X)$ is implemented with 4 adders/subtractors, 6 tables and a 6-operand adder. The tables a_0 to a_4 in Fig. 2-5(a) that implement the first 5 terms of $P_{ATA}(X)$ are significantly large. They contribute to 97.17% of the total memory size, $N_t = 868352$ bits. The proposed iATA method in Chapter 5 attempts to reduce their size or even remove some of them.

2.2.2 Symmetric Bipartite Table Method (SBTM)

The SBTM method by Schulte and Stine [Sch99] is the generalization of the initial bipartite method for computing the reciprocal function by Das Sarma and Matula [Das95]. In SBTM, X is divided into three parts as shown in Fig. 2-5(b), where $\lambda_i = 2^{-n_i}$ and $X_i = [0, 1 - 2^{-n_i}]$ is n_i bits in length for $i = \{0, 1, 2\}$. Based on Taylor series approximation, a function $f(X)$ is first approximated as follows:

$$\begin{aligned} f(X) &= f(X_0 + \lambda_0 X_1 + \lambda_0 \lambda_1 X_2) \\ &= f(X_0 + \lambda_0 X_1 + \delta_2 + \lambda_0 \lambda_1 X_2 - \delta_2) \\ &\approx D_0^{SBTM} + D_1^{SBTM} \end{aligned}$$

where

$$\begin{aligned} D_0^{SBTM} &= f(X_0 + \lambda_0 X_1 + \delta_2) \\ D_1^{SBTM} &= f^{(1)}(X_0 + \lambda_0 X_1 + \delta_2)(\lambda_0 \lambda_1 X_2 - \delta_2) \end{aligned}$$

and $\delta_2 = \frac{1}{2} \lambda_0 \lambda_1 (1 - 2^{-n_2})$ is the mid-value between the minimum and maximum values of $\lambda_0 \lambda_1 X_2$. For saving memory, SBTM replaces the term $\lambda_0 X_1$ of D_1^{SBTM} by its mid-value $\delta_1 = \frac{1}{2} \lambda_0 (1 - 2^{-n_1})$. Therefore, the approximation equation becomes:

$$\begin{aligned} P_{SBTM} &= f(X_0 + \lambda_0 X_1 + \delta_2) + \\ &\quad f^{(1)}(X_0 + \delta_1 + \delta_2)(\lambda_0 \lambda_1 X_2 - \delta_2) \\ &= a_0(X_0, \lambda_0 X_1) + a_1(X_0, \lambda_0 \lambda_1 X_2). \end{aligned} \tag{2.35}$$

P_{SBTM} is implemented with 2 tables a_0 and a_1 (Fig. 2-5(b)), a 2-operand adder and some control logics. It capitalizes on the symmetrical property of the table entries for $a_1(X_0, \lambda_0 \lambda_1 X_2)$ as illustrated below. Taking $2\delta_2 - \lambda_0 \lambda_1 X_2$ (equal to one's complement of X_2 , i.e. $\overline{X_2}$) as the input to table a_1 , the symmetrical property is observed as follows:

$$\begin{aligned} a_1(X_0, 2\delta_2 - \lambda_0 \lambda_1 X_2) &= f^{(1)}(X_0 + \delta_1 + \delta_2)(2\delta_2 - \lambda_0 \lambda_1 X_2 - \delta_2) \\ &= -f^{(1)}(X_0 + \delta_1 + \delta_2)(\lambda_0 \lambda_1 X_2 - \delta_2) \\ &= -a_1(X_0, \lambda_0 \lambda_1 X_2). \end{aligned}$$

Thus, the table $a_1(X_0, \lambda_0 \lambda_1 X_2)$ is symmetrical at its mid-point and needs to store only the values corresponding to those X_2 with '0' in its MSB. For X_2 with '1' in its MSB, the bits of $\overline{X_2}$ other than the MSB are used to address the table. The output value of the table is then negated. As such, only half of the original table is needed. With some mathematical manipulations, the symmetrical property is integrated in the proposed iATA as described in Chapter 5.

2.2.3 Multipartite Table Method (MTM)

Stine and Schulte suggested in [Sti99] that X_2 in P_{SBTM} (2.35) can be decomposed into more than one parts. In their proposed STAM (Symmetric Table Addition Method), X_2 is divided into m parts as shown in Fig. 2-5(c), where $\lambda_i = 2^{-n_i}$, $X_i = [0, 1 - 2^{-n_i}]$ is n_i bits in length, $i = \{0, 1, 2, \dots, h\}$ and $h = \{1, 2, \dots, m\}$. Maintaining the symmetrical property, (2.35) is modified to form the STAM approximation equation as follows:

$$P_{STAM} = f(X_0 + \lambda_0 X_1 + \delta_2) + \sum_{h=1}^m [f^{(1)}(X_0 + \delta_1 + \delta_2)(\lambda_0 \lambda_1 \prod_{j=0}^{h-1} \lambda_{2,j} X_{2,h} - \delta_{2,h})] \quad (2.36)$$

$$= a_0(X_0, \lambda_0 X_1) + \sum_{h=1}^m a_h(X_0, \lambda_0 \lambda_1 \prod_{j=0}^{h-1} \lambda_{2,j} X_{2,h}) \quad (2.37)$$

where $\lambda_{2,0} = 1$ is omitted from Fig. 2-5(c). As such, the large table $a_1(X_0, \lambda_0 \lambda_1 X_2)$ in SBTM is split into m smaller tables. STAM was first improved by Muller [Mul99], who proposed to use different wordlengths of X_0 for different h tables, i.e. X_0 (wordlength n_0) is replaced with $X_{0,h}$ (wordlength $n_{0,h}$) for tables $a_1 - a_m$ in (2.37). Subsequently, STAM and Muller's multipartite method were generalized by de Dinechin and Tisserand with a geometrical approach in the unified MTM [deD05] as summarized below. The derivative $f^{(1)}(X_0 + \delta_1 + \delta_2)$ in (2.36) is a gradient g . The methods in [Sti99] and [Mul99] are based on Taylor series approximation and their g values are determined by the values of the first derivatives of the function. On the other hand, the g values in the unified MTM [deD05] are determined via geometrical approach. With the approximation equation

$$P_{MTM} = f(X_0 + \lambda_0 X_1 + \delta_2) + \sum_{h=1}^m g(X_{0,h})(\lambda_0 \lambda_1 \prod_{j=0}^{h-1} \lambda_{2,j} X_{2,h} - \delta_{2,h})$$

and given a decomposition $D = \{n_2, n_{0,h}, n_{2,h}, m\}$, the gradient $g(X_{0,h})$ is adjusted such that the minimum approximation error, $f(X) - P_{MTM}$, is obtained. This process is repeated for all the possible combinations of $n_2, n_{0,h}$ and $n_{2,h}$ with a given m value. The decompositions that fulfill the error budget are kept. This geometrical approach provides tighter error bounds compared to Taylor series approximation, which gives only the upper bounds. This advantage is translated to

higher possibility for memory reduction by the unified MTM when compared to STAM and the Muller's multipartite method.

2.2.4 Summary on Accuracy

The tables-and-additions methods (and other LUT-based methods) aim to achieve faithful rounding computations, i.e., the maximum absolute error of the computations of a function is less than one unit in the last place (ulp), 2^{-n} , for a n -bit input/output (IO) wordlength. Table 2-3 shows the various elementary functions computed for the IO wordlengths of 16, 20 and 24 bits by the tables-and-additions methods described in Section 2.2.1-2.2.3. For example, ATA was used to compute the reciprocal, square root, exponential, sine, natural logarithm and arc tangent functions with 24-bit IO wordlength. Although not all the methods are presented with the results for all elementary functions for all the three wordlengths, the tables-and-additions methods, in general, can be extended with certain effort to other elementary functions (including those not listed in Table 2-3) for other IO wordlengths (accuracy). For example, in the study in [Hsi08], the multipartite method [deD05] is used to compute functions such as square root and binary logarithm with IO wordlengths ranging from 8 bits to 24 bits at the step size of 4 bits.

TABLE 2-3: ELEMENTARY FUNCTIONS COMPUTED FOR VARIOUS WORDLENGTH

Functions	Reciprocal			Square Root			Exponential			Sine			Natural Logarithm			Binary Logarithm			Arc-tangen		
Wordlength (bits)	16	20	24	16	20	24	16	20	24	16	20	24	16	20	24	16	20	24	16	20	24
ATA [Won95]	-	-	✓	-	-	✓	-	-	✓	-	-	✓	-	-	✓	-	-	-	-	-	✓
SBTM [Sch99]	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	✓	✓	✓	-	-	-
STAM [Sti99]	✓	-	✓	✓	-	✓	✓	-	✓	✓	-	✓	✓	-	✓	-	-	-	-	-	-
MTM [deD05]	✓	-	✓	-	-	-	✓	-	✓	✓	-	✓	-	-	-	-	-	-	-	-	-

2.3 Summary

The two main categories of the logarithmic conversion methods, Mitchell-based and tables-and-additions, that are efficient (fast) in computation, are discussed in Section 2.1 and 2.2. The Mitchell-based methods are area and speed efficient but incur conversion errors. The highest existing accuracy obtained is less than 12 bits. On the other hand, the tables-and-additions methods cater for high accuracy (up to 24 bits) conversion and are efficient (fast) among the high accuracy methods but incur large area cost.

CHAPTER 3. Unified Mitchell-based Approximation for Efficient Logarithmic Conversion

3.1 Overview

As the existing Mitchell-based methods are not systemically optimized and/or non-scalable as discussed in Sections 1.1 and 2.1, this work aims to develop a systematical approach to achieve both hardware optimization and accuracy scalability. After studying carefully and extensively the existing Mitchell-based methods, five design parameters that impact both the accuracy and hardware merits of an approximation function are identified. Specifically, the parameters are:

- a) T = number of MSBs of m used for approximation function;
- b) P = maximum number of power-of-two terms (POTTs) in each of the gradient values (a_r of (2.7) or C_r of (2.8));
- c) V = minimum non-zero value allowed for the POTTs;
- d) R = number of regions of the approximation;
- e) B = number of MSBs of m used to divide the Mitchell range M .

Theoretically, the number of possible values for the design parameters is large as given in Table 3-1 for $n = 32$. Besides, for a given pair of P and V , there are many possible combinations and choices of POTTS for the gradient of the approximation straight line in each region. In addition, for a given pair of R and B , there are many possible combinations of regions that the Mitchell range $M = [1, 2)$ can be divided into. Therefore, the possible combinations of all the parameters are numerous. It is impractical to sweep all the ranges to find an optimized approximation function that meets a desired accuracy requirement. No existing approach can be found in the literature for the exploration of such a huge design space of the five design parameters. In this work, a novel

TABLE 3-1: POSSIBLE RANGE FOR DESIGN PARAMETERS ($n = 32$)

Design Parameters		Possible Values
No. of MSBs of m used	T	$0, 1, 2, \dots, (n - \log_2 n)$
Max. No. of POTTs	P	$0, 1, \dots, T - 1$
Min. Value of POTTs	V	$2^{-1}, 2^{-2}, \dots, 2^{-(T-1)}$
No. of Regions	R	$1, 2, \dots, 2^{31}$
No. of Bits to Select Region	B	$\lceil \log_2 R \rceil, \lceil \log_2 R \rceil + 1, \dots, 31$

method, named the Unified Mitchell-based Approximation (UMA), is proposed to systematically explore the design space in order to find an optimized approximation circuit fulfilling a given accuracy requirement. In the proposed method, the range of the design parameters is constrained and a new systematical search algorithm is introduced for sweeping the design parameter values within the constrained ranges. The proposed method is presented in the following sections, with Section 3.2 discussing the architectural model and the impact of the design parameters on the accuracy and hardware complexity of the approximation function, and Sections 3.3 – 3.8 describing in details the proposed search algorithm with the proposed constraints on the range of values of the design parameters.

3.2 Architectural Model and Design Parameters

In the proposed method, the general approximation function for a R -region approximation is formulated as:

$$\begin{aligned} & \log_2(1 + m) \\ & \cong \text{LA}_{R,\#}(1 + m) \\ & = m + C_r \cdot m_{MSB}^* + D_r, \quad r \in \{1, 2, \dots, R\} \end{aligned} \quad (3.1)$$

$$= \begin{cases} m + C_1 \cdot m_{MSB}^* + D_1 \\ m + C_2 \cdot m_{MSB}^* + D_2 \\ \vdots \\ m + C_R \cdot m_{MSB}^* + D_R \end{cases} \quad (3.2)$$

$$= \begin{cases} m + (c_{1.1} + c_{1.2} + \dots + c_{1.P}) \cdot m_{MSB}^* + D_1 \\ m + (c_{2.1} + c_{2.2} + \dots + c_{2.P}) \cdot m_{MSB}^* + D_2 \\ \vdots \\ m + (c_{R.1} + c_{R.2} + \dots + c_{R.P}) \cdot m_{MSB}^* + D_R \end{cases} \quad (3.3)$$

where $m_{MSB}^* \in \{m_{MSB}, \bar{m}_{MSB}\}$, m_{MSB} is the MSBs of m , $\bar{m}_{MSB}$ is the 1's complement of m_{MSB} , $C_r = c_{r.1} + c_{r.2} + \dots + c_{r.P}$ and D_r are respectively the gradient and constant of the approximation function in the r -th region g_r , $c_{r.p}$ ($p = \{1, 2, \dots, P\}$) are the POTTs contained in the gradient C_r . In the term $c_{r.p} \cdot m_{MSB}^*$, $m_{MSB}^* = m_{MSB}$ if $c_{r.p}$ is positive. Otherwise, $m_{MSB}^* = \bar{m}_{MSB}$ and the negative sign of $c_{r.p}$ is omitted. The number of MSBs used for m_{MSB} or $\bar{m}_{MSB}$ is determined such that every $c_{r.p} \cdot m_{MSB}^*$ is truncated at the 2^{-T} binary position for non-zero $c_{r.p}$, i.e. the bits smaller than 2^{-T} are discarded. For example, if $c_{r.p} \cdot m_{MSB}^* = 2^{-3} \cdot m_{MSB}^*$ and $T = 8$, the number of MSBs for m_{MSB}^* is $T - (-\log_2(|c_{r.p}|)) = 8 - (-\log_2(2^{-3})) = 5$. The constants D_r are also truncated at the 2^{-T} binary position. For illustration, the approximation function of

Fig. 3-1: Generic architectural model for approximation

the 3-region approximation (3.4) by Abed and Siferd [Abe03] is expressed as (3.5) according to the form of (3.3).

$$\begin{aligned}
 & \text{LA}_{3.\text{Abed}}(1 + m) \\
 &= \begin{cases} m + (2^{-2}) \cdot m_{\text{MSB}4} & m \in [0, 0.25) \\ m & m \in [0.25, 0.75) \\ m + (2^{-2}) \cdot \bar{m}_{\text{MSB}4} & m \in [0.75, 1) \end{cases} \quad (3.4) \\
 &= \begin{cases} m + (2^{-2}) \cdot m_{\text{MSB}}^* + 0 & m \in [0, 0.25) \\ m + (0) \cdot m_{\text{MSB}}^* + 6 \cdot 2^{-6} & m \in [0.25, 0.75) \\ m + (-2^{-2}) \cdot m_{\text{MSB}}^* + 0 & m \in [0.75, 1) \end{cases} \quad (3.5)
 \end{aligned}$$

For their design, there are 3 regions (R) for approximation, 2 MSBs (B) of m for region selection and only 6 MSBs (T) of m used for approximation. The maximum number (P) of POTTs is 1 and the minimum non-zero value (V) of the POTTs is 2^{-2} . In a word, $R = 3$, $B = 2$, $P = 1$, $V = 2^{-2}$ and $T = 6$. Note that although $P = 1$ in this design, the number of POTTs in a region can be less than P (no POTT in the second region in this case).

The general approximation function (3.3) is modeled by the generic approximation circuit architecture shown in Fig. 3-1. In (3.3), the term $(c_{r,1} + c_{r,2} + \dots + c_{r,p}) \cdot m_{\text{MSB}}^*$ is implemented with only shift-and-add operations as in the Mitchell-based methods. As such, in each r -th region g_r , there are $P + 2$ operands of addition: m , $c_{r,1} \cdot m_{\text{MSB}}^*$, $c_{r,2} \cdot m_{\text{MSB}}^*$, ..., $c_{r,p} \cdot m_{\text{MSB}}^*$ and D_r . To reduce hardware, the $(P + 2)$ -operand addition in each region is performed in the same adder (adder tree of Fig. 3-1), i.e., the operands from each region are multiplexed. One of the operands $c_{r,p} \cdot m_{\text{MSB}}^*$ ($r \in \{1, 2, \dots, R\}$) is selected by SEL_p ($p = \{1, 2, \dots, P\}$) depending on the region in

which the input falls. The selected value I_p is added with the constant D_r of the same region. The different operands, $c_{r,p} \cdot m_{MSB}^*$ and D_r , for different regions are selected by the multi-input MUXes, MUX_p and MUX_D , as shown in Fig. 3-1. In this work, the $(P + 2)$ -operand addition is modeled with the well-known Dadda's tree adder [Dad65] followed by a simple ripple carry adder (RCA) whereas the multi-input MUXes (MUX_p and MUX_D) are modeled with 2-to-1 MUXes. The wordlength of I_p is $h_p = T - \min_{\forall r} \{-\log_2 |c_{r,p}|\}$. For m , only T MSBs are inputted into the adder tree because all the other operands are truncated at 2^{-T} binary position. The bits of m from binary position $2^{-(T+1)}$ to 2^{-27} are directly concatenated at the output as shown in Fig. 3-1. Note that the worst case value of T is used for the wordlength of D_r , and hence D too, in the circuit.

Different choices of values of the design parameters result in different approximation function (3.3), consequently, different accuracy and conversion circuit. The impact of the design parameters on the accuracy and hardware merits are discussed in the following subsections, assuming that when a parameter is discussed, the other parameter values are fixed.

3.2.1 Impact on Accuracy

The impact of each of the five parameters on the accuracy is analyzed and discussed here.

T is the number of MSBs of m used for approximation function. The value of T determines the binary position 2^{-T} at which the operands, I_p ($p = \{1, 2, \dots, P\}$) and D , are truncated and, hence, each of the operands contributes to a maximum truncation error of 2^{-T} . Increasing the value of T reduces the truncation error but this does not necessarily lead to higher accuracy because the larger truncation errors due to smaller T values could be beneficial in cancelling out the approximation error, ε (3.6), leading to more effective approximation function.

$$\varepsilon = \log_2(1 + m) - (m + C_r \cdot m^* + D_r) \quad r \in \{1, 2, \dots, R\} \quad (3.6)$$

where $m^* \in \{m, \bar{m}\}$, $\bar{m}$ is the 1's complement of m and, unlike in (3.3), D_r here is not truncated.

P is the maximum number of POTTs, $c_{r,p}$ ($p = \{1, 2, \dots, P\}$) in (3.3), that can be contained in the gradient C_r in (3.1). The larger the value of P , the more choices of slope the piecewise straight lines (3.1) have and vice versa. Increasing P value helps to improve accuracy because there are more straight lines to be chosen for better fitting the non-linear $\log_2(1 + m)$ curve.

V is the minimum magnitude of a POTT contained in the gradient C_r in (3.1). Given a V value, e.g. $V = 2^{-3}$, a POTT may take any value of $-2^{-1}, -2^{-2}, \dots, -V, V, \dots, 2^{-2}$, or 2^{-1} . Decreasing

the V value increases the choices of value that can be adopted as a POTT in the gradient C_r . In other words, there are more choices of slopes for the piecewise straight lines and, hence, more choices of straight lines for better fitting the $\log_2(1 + m)$ curve.

R is the number of regions for approximation function (3.3). The larger the value of R , the smaller the regions that the Mitchell range $M = [1,2)$ is divided into and, the smaller curvature the $\log_2(1 + m)$ curve has within each region. Consequently, increasing the value of R helps to improve accuracy because the piecewise straight lines within the regions deviate less from the $\log_2(1 + m)$ curve. It is worth noting that, given a set of values for the design parameters T , P , V and B , there is always a minimum value of R such that an approximation function (3.3) meets the desired accuracy requirement, and below this minimum value, the desired accuracy cannot be fulfilled.

B is the number of MSBs of m used to divide the Mitchell range M into regions and has a minimum value of $B_{min} = \lceil \log_2(R) \rceil$. Any B value larger than $\log_2(R)$ allows non-uniform regions. For example, if $B = 2$ is used for $R = 2$ ($B_{min} = 1$), the two regions, g_1 and g_2 , of the Mitchell range $M = [1,2)$ can be $\{g_1, g_2\} = \{[1,1.25), [1.25,2)\}$, $\{[1,1.5), [1.5,2)\}$ or $\{[1,1.75), [1.75,2)\}$. Given a R value, increasing B value to allow non-uniform regions helps to improve the accuracy because the segment where $\log_2(1 + m)$ has larger curvature can be divided into smaller regions while the part with smaller curvature divided into larger regions.

3.2.2 Impact on Hardware Merits

In this subsection, the model for the evaluation of hardware merits is first described before the impact of the design parameters on the hardware merits is discussed. In this work, the hardware merits, such as area cost and critical path delay, of the approximation circuit shown in Fig. 3-1 are evaluated based on the unit-gate model [Low12a], [Ver02]. In this model, a 2-input monotonic gate, such as a NAND/AND gate, has one unit of area and one unit of delay. The area (A) and delay (τ) of a full adder (FA), half adder (HA) and 2-to-1 multiplexer (MUX) are given in Table 3-2. In the hardware evaluation, the combinational control logic block in Fig. 3-1 is omitted.

The total area cost is the sum of the area cost of the adder tree and MUXes. The area cost of the adder tree is the total area cost incurred by all the FAs and HAs required in the Dadda's tree adder

TABLE 3-2: AREA AND DELAY IN UNIT-GATE MODEL

Cells	NAND	AND	NOT	MUX	HA	FA
Area (A)	1	1	0	2	3	7
Delay (τ)	1	1	0	2	2	4

[Dad65] and, hence, is a function of the number of operands (m_{MSBT} , I_p ($p = \{1, 2, \dots, P\}$) and D) and their wordlengths. For the multi-input MUXes, MUX_p and MUX_D , the area cost is the total cost incurred by all the 2-to-1 MUXes used to implement the MUXes. For a v -input MUX with each input wordlength of w , the number of 2-to-1 MUXes needed is $w \times (v - 1)$.

The overall critical path delay is the sum of the critical path delays of the adder tree and the multi-input MUXes. The critical path of the adder tree depends on the number of stages needed for reducing the height of the tree to two and the length of the RCA [Dad65]. Each reduction stage has a delay of a FA whereas an i -bit RCA has a propagation delay of $i - 1$ FAs and a HA. For the multi-input MUXes, the critical path is within the one with most inputs. Implemented with 2-to-1 MUXes, a v -input MUX has $\lceil \log_2(v) \rceil$ 2-to-1 MUXes in the critical path.

The impact of the design parameters on hardware merits is discussed as followed. Increasing the T value increases the length of the operands, m_{MSBT} , I_p ($p = \{1, 2, \dots, P\}$) and D . This in turn increases the area and delay of the adder tree. However, it increases only the area but not the delay of the MUXes. It is important to highlight that T has the most significant impact on the overall critical path delay because the length of the RCA in the adder tree equals to T and the delay of the RCA is the dominant component due to the propagative effect.

If the P value is increased, the number of operands, I_p , to the adder tree increases too. This in turn increases the area and delay of the adder tree. Depending on the POTTs, $c_{r,p}$, used in each region, increasing P may or may not increase the area and delay incurred by the MUXes. For the worst scenario that none of the regions has common POTTs, $c_{r,p}$, the MUXes are the largest. On the other hand, for the best scenario when all the regions have the same POTTs, the MUXes are not needed at all.

When the R value is increased, the above two extreme scenarios are also applicable. For the worst scenario, increasing R will increase the number of inputs to the MUXes and, hence, their area and delay. For the best scenario, increasing R has no effect on the MUXes. However, both the worst and best scenarios are unlikely to happen. Thus, increasing the R value may or may not increase the area and delay of the MUXes depending on the POTTs of all the regions. The value of R has no impact on the area and delay of the adder tree.

TABLE 3-3: IMPACT OF DESIGN PARAMETERS ON ACCURACY AND COMPLEXLY

Design parameters	Accuracy	Complexity			
		Adder Tree		MUX	
		Area	Delay	Area	Delay
$T \uparrow$	$\leftrightarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\times$
$P \uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\leftrightarrow$	$\leftrightarrow$
$V \downarrow$	$\uparrow$	$\times$	$\times$	$\leftrightarrow$	$\leftrightarrow$
$R \uparrow$	$\uparrow$	$\times$	$\times$	$\leftrightarrow$	$\leftrightarrow$
$B \uparrow$	$\uparrow$	$\times$	$\times$	$\times$	$\times$

The impact of V on hardware merits is similar to that of R . If V value is decreased, there are more choices of value for each POTT. As such, the POTTs in a region are more likely to be different from the POTTs in other regions, resulting in more inputs to the multi-input MUXes and, hence, larger area and delay of the MUXes.

Unlike the other four parameters, increasing B value has no impact on the area cost and critical path delay of both the adder tree and MUXes but will increase only the complexity of the control logic for generating the signals, SEL_p and SEL . The impact is negligible. However, the accuracy is unlikely to be satisfied by increasing B value alone if R is not sufficiently large.

The impact of the design parameters on accuracy and hardware complexity is summarized in Table 3-3, where $\uparrow$, $\downarrow$, $\leftrightarrow$ and $\times$, respectively, denote that the value of a quantity increases, decreases, may increase/decrease and not affected. It is worth to mention that, given an accuracy requirement, increasing the value of a design parameter may not lead to worse hardware merits as it may allow the other design parameter values to be reduced, resulting in better hardware merits. Therefore, the parameters cannot be optimized individually. Consequently, the design space consisting of all the combinations of the parameters is huge and poses a challenge for finding a set of parameters that constitutes an optimized approximation circuit for a given accuracy requirement. The practicability of the ranges of the parameters is analyzed, the search is confined within practical and feasible ranges of the parameters and a systematical search algorithm is proposed as presented in the subsequent sections.

3.3 Design Space Delimitation

After analyzing the ranges of the five parameters, it is proposed in this research work to constrain the ranges to practical and reasonable values so that the design space is significantly reduced but at the same time an optimized design (approximation circuit) can still be found.

The parameter T , which is the number of MSBs of m used for approximation, is first considered. T is also used for truncating the operands $c_{r,p} \cdot m_{MSB}^*$ and D_r (3.3). If an operand is truncated at 2^{-T} position, a maximum truncation error of 2^{-T} is incurred. Excluding all other errors, the truncation of a single operand causes the approximation of $\log_2(1+m)$ by the approximation function, $LA_{R,\#}(1+m)$, to incur an error percentage $EP = \frac{\log_2(1+m) - LA_{R,\#}(1+m)}{\log_2(N)} \cdot 100 < \frac{2^{-T}}{\log_2(N)} \cdot 100$. The *MAEP* due to the truncation will occur at the minimum value of $\log_2(N)$, i.e., $\log_2(2) = 1$, where $MAEP < 2^{-T} \cdot 100$ ($T < -\log_2(\frac{MAEP}{100})$). Therefore, it is proposed to constrain T such that $T_0 - 3 \leq T \leq T_0 + 4$, where

$$T_0 = \lfloor -\log_2(\frac{Acc}{100}) \rfloor \quad (3.7)$$

and $Acc \approx 2^{-T_0} \cdot 100$ is the required accuracy expressed in term of *MAEP*. Hence, if an operand is truncated at the 2^{-T_0} position, the maximum truncation error leads to *MAEP* approximately equals to *Acc*. The maximum total truncation errors could be larger if more than one operands (I_p ($p = \{1, 2, \dots, P\}$) and D) are truncated. However, the accuracy does not always suffer from the truncation error as the truncation error might cancel out the approximation errors. At the upper bound of T , $T_0 + 4$, the maximum truncation error leads to a *MAEP* of about one-sixteenth of *Acc* ($2^{-T_0-4} \cdot 100 = \frac{1}{16} 2^{-T_0} \cdot 100 \approx \frac{1}{16} Acc$), which is negligible compared to the approximation error. In other words, the upper bound represents the extreme end where the *MAEP* is solely due to approximation error. On the other hand, the lower bound, $T_0 - 3$, may lead to *MAEP* about eight times of *Acc* and represents the other extreme end where the truncation error becomes significant in the *MAEP* and any T value less than $T_0 - 3$ would result in even larger error, hindering the accuracy requirement from being satisfied.

With T constrained, the maximum number of POTTs, P , can be varied between $[0, T - 1]$ and the minimum value of POTTs is fixed at $V = 2^{-(T-1)}$, as any POTT less than $2^{-(T-1)}$ would be immaterial due to truncation.

As discussed in Section 3.2A, given a set of values for T , P , V and B , the parameter R is important to enable an approximation to meet the accuracy requirement and, hence, it is not constrained, i.e., it can be any positive integer value. However, as the pairs of R and B values could lead to numerous possible combinations of regions that M is divided into, it is proposed in this research work to constrain B by determining its value from R as discussed in Section 3.5.

The design space is still large despite the parameters are constrained. In addition, even with the ranges of P and V constrained, the number of possible combinations of POTTs that can be used as the gradients of the regions is also large. This research work introduces a systematical search method to explore the design space as follows.

3.4 Design Space Exploration

As the range of T is fixed, its entire range is swept in the exploration of the design space in order to find the optimized solution. A solution here is referred to as an approximation function that satisfies the accuracy requirement. For each T , P starts from 0 and is increased by 1 until the solution found is not better in term of hardware merits than the one found with the previous P value. Within each P , the Mitchell range M is divided until a solution is found. In order words, R starts from 1 and is incremented by 1 until a solution is found. The search sequence during the design space exploration is better visualized in Fig. 3-2, where the parameters B and V are not swept as B is determined when R is fixed (Section 3.5) and V is limited by T . Note that the well-known branch-and-bound searching method cannot be applied on the tree in Fig. 3-2. A solution found in one branch does not guarantee there are no better solutions existed in the branches to the right (higher parameter values) as discussed in Section 3.3.

With the above searching sequence and a given accuracy requirement Acc , the initial values for the parameters are set at $R = 1$, $B = 0$, $P = 0$, $V = 2^{-(T_0-4)}$ and $T = T_0 - 3$, where T_0 is given in (3.7). With $R = 1$ and $P = 0$, i.e. one approximation region and no POTT for the gradient value, the only possible gradient value of the only approximation straight line (approximation function) is 0. The initial approximation function is expressed as

$$LA_{1,a}(1 + m) = m + D_1 \quad m \in [0,1) \quad (3.8)$$

where D_1 is a constant value truncated at 2^{-T} position. Error analysis (Section 3.7) is performed

Fig. 3-2: Design space tree of the proposed UMA

for the only region $g_1 = [0,1)$ to determine whether the approximation function (3.8) is able to meet the accuracy requirement Acc . If not, the region is divided, i.e. R is incremented, in the way described below. As mentioned in Section 2.1.1, the subscription ‘1’ in $LA_{1.a}(1+m)$ represents the value of R and ‘a’ is a unique letter for the approximation function (3.8).

3.5 Division of Regions

If the approximation function of a region fails to meet the accuracy requirement, the region is referred to as a “fail” region (f-region) and it is divided equally into two new regions for searching new approximation functions. For example, if the region in (3.8) is a f-region, it is divided into two regions $g_1 = [0,0.5)$ and $g_2 = [0.5,1)$ and a new 2-region approximation function is obtained below.

$$LA_{2.b}(1+m) = \begin{cases} m + D_1 & m \in [0,0.5) \\ m + D_2 & m \in [0.5,1) \end{cases}$$

The number of regions is now $R = 2$ and it requires $B = 1$ MSBs of m to identify the two regions. In general, $B = -\log_2 W_{min}$, where W_{min} is the size of the smallest region. Error analysis is performed for each of the two new regions. Without loss of generality, assume that accuracy requirement is satisfied in g_2 and it is referred to as a “success” region (s-region). The f-region g_1 is again divided equally into two new regions, $g_1 = [0,0.25)$ and $g_2 = [0.25,0.5)$. In general, only f-regions are divided while s-regions are retained with their region numbers rearranged in sequence. E.g., the previous $g_2 = [0.5,1)$ now becomes $g_3 = [0.5,1)$. Consequently, the following new 3-region approximation function is obtained.

$$LA_{3.c}(1+m) = \begin{cases} m + D_1 & m \in [0,0.25) \\ m + D_2 & m \in [0.25,0.5) \\ m + D_3 & m \in [0.5,1) \end{cases} \quad (3.9)$$

While the other design parameter values remain unchanged, R and B are now $R = 3$ and $B = 2$. Error analysis is performed for the two new regions, g_1 and g_2 , only. If the accuracy requirement is now met in both regions, a solution (valid approximation function) with $R = 3$ is found. In general, for a given set of T and P values, R is increased by dividing the f-regions until a solution is found, i.e., Bound2 in Fig. 3-2 is reached. In other words, given a fixed number of POTTs (P) in the gradient values of an approximation function, the proposed search algorithm divides the Mitchell range M into sufficiently small regions such that the approximation of the $\log_2(1+m)$ by the approximation function is able to meet the desired accuracy in each and every region.

Fig. 3-3: Divisions of regions

The division of regions is illustrated graphically with an example in Fig. 3-3, where “✓” and “x” respectively denote s-region and f-region. Note that Fig. 3-3 is not the same as the one in the above discussion. The region g_2 in Step (3) of Fig. 3-3 is assumed to be a f-region and the final values are $R = 4$ and $B = -\log_2 \frac{1}{8} = 3$.

The solution(s) found for a P value is(are) evaluated (Section 3.8) and compared to determine the best one for the P value (in one P branch of Fig. 3-2). In the above discussion, only one solution, $LA_{3.c}(1+m)$ (3.9), is found for $P = 0$. If $P = 0$ or the solution has better hardware merits than the one found for the previous P value, then the solution is recorded, P is increased, and R starts from 1 again and is increased by dividing the f-regions until another solution is found. Otherwise, the search moves on to next T value. When $P = 0$, for each of the new regions divided from f-regions, there is only one approximation straight line to be considered for error analysis due to only one possible gradient value, zero. On the other hand, for $P > 0$, there are more than one possible gradient values and, hence, more than one approximation straight lines for each new region as described in next section.

3.6 Selection of POTTs

When $P = 0$, there is no POTT used for the gradient i.e. the only gradient is 0. When $P > 0$, there are one or more POTTs that can be used for the gradients. For example, with a search in the branch (Fig. 3-2) where $T = T_0 - 3$, $P = 1$ and $R = 1$, which gives $B = 0$ and $V = 2^{-(T_0-4)}$, the set of all possible choices of POTTs is

$$S_{POTT} = \{-2^{-1}, \dots, -2^{-(T_0-4)}, +2^{-(T_0-4)}, \dots, +2^{-1}\} \quad (3.10)$$

and the 1-region approximation function will be

$$LA_{1.d}(1+m) = m + C_1 \cdot m_{MSB}^* + D_1 \quad m \in [0,1) . \quad (3.11)$$

TABLE 3-4: AN EXAMPLE OF C_r FOR APPROXIMATION FUNCTION (5.3)

C_1	2^{-1}	2^{-5}	2^{-6}
C_2	2^{-1}	2^{-4}	–

With $P = 1$, each gradient can have not more than one POTT and any POTT in S_{POTT} can be used as the value of C_1 . To select a POTT for C_1 , error analysis (Section 3.7) is performed on (3.11). If no solution can be found with any of the POTTs in S_{POTT} , the region $g_1 = [0,1)$ is a f-region and to be divided. As an example, assume that solutions are found with $R = 2$ as expressed in (3.12) in which each region has more than one suitable POTTs.

$$\begin{aligned}
 & \text{LA}_{2,e}(1+m) \\
 &= \begin{cases} m + C_1 \cdot m_{MSB}^* + D_1 & m \in [0,0.5) \\ m + C_2 \cdot m_{MSB}^* + D_2 & m \in [0.5,1) \end{cases} \\
 &= \begin{cases} m + c_{1,1} \cdot m_{MSB}^* + D_1 & m \in [0,0.5) \\ m + c_{2,1} \cdot m_{MSB}^* + D_2 & m \in [0.5,1) \end{cases} \quad (3.12)
 \end{aligned}$$

where the POTTs of $C_r = c_{r,1}$ ($r = \{1,2\}$) are assumed to have the suitable values shown in Table 3-4 with three valid values for C_1 and two for C_2 . In this case, a solution for (3.12) can be obtained by selecting one of three values of C_1 and one of two values of C_2 . In other words, there are $3 \times 2 = 6$ solutions, $\{C_1, C_2\} = \{2^{-1}, 2^{-1}\}, \{2^{-1}, 2^{-4}\}, \{2^{-5}, 2^{-1}\}, \{2^{-5}, 2^{-4}\}, \{2^{-6}, 2^{-1}\}$ and $\{2^{-6}, 2^{-4}\}$. The hardware merits of the six solutions are evaluated (Section 3.8) and compared to determine the best solution for the P value ($P = 1$ in this case). Note that in this example, $\{C_1, C_2\} = \{2^{-1}, 2^{-1}\}$ would be the best choice as no MUX is needed.

For larger P , there are more possible gradient values for selection. Given a pair of P and $V = 2^{-(T-1)}$, a gradient value may contain zero, one, two, ... or P POTTs from the set S_{POTT} (3.10). To find the best solution, it is necessary to determine all possible combinations. Let $C = c_1 + c_2 + \dots + c_{P'}$, denoted as $\{c_1, c_2, \dots, c_{P'}\}$, be a possible gradient value that can be used as C_r , where P' is the number of POTTs and can have a value between 0 to P , i.e. $P' = \{0, 1, 2, \dots, P\}$. If $P' = 0$, $C = 0$ is the only choice. For $P' \geq 1$, there are $N_V C_{P'}$ possible combinations for $\{c_1, c_2, \dots, c_{P'}\}$, where $N_V = 2(-\log_2 V)$ is the number of possible POTTs available for selection, i.e., the size of S_{POTT} . In general, the total number of possible combinations for $\{c_1, c_2, \dots, c_{P'}\}$, given the values of P and V , is $\sum_{P'=0}^P N_V C_{P'}$. However, some of them contain both the POTTs 2^{-i} and -2^{-i} ($i = \{1, 2, \dots, -\log_2 V\}$) and are considered to be invalid. For illustration, Table 3-5 lists the valid combinations for $\{c_1, c_2, \dots, c_{P'}\}$ ($P' = \{0, 1, 2\}$) when $P = 2$ and $V = 2^{-3}$. The invalid combinations, $\{2^{-1}, -2^{-1}\}, \{2^{-2}, -2^{-2}\}$ and $\{2^{-3}, -2^{-3}\}$, are excluded from the list.

TABLE 3-5: POSSIBLE GRADIENT VALUES ($P = 2, V = 2^{-3}$)

P'	Possible Combinations of $\{c_1, c_2\}$			
0	0,0			
1	$2^{-1}, 0$		$-2^{-1}, 0$	
	$2^{-2}, 0$		$-2^{-2}, 0$	
	$2^{-3}, 0$		$-2^{-3}, 0$	
2	$2^{-1}, 2^{-2}$	$2^{-1}, -2^{-2}$	$-2^{-1}, 2^{-2}$	$-2^{-1}, -2^{-2}$
	$2^{-1}, 2^{-3}$	$2^{-1}, -2^{-3}$	$-2^{-1}, 2^{-3}$	$-2^{-1}, -2^{-3}$
	$2^{-2}, 2^{-3}$	$2^{-2}, -2^{-3}$	$-2^{-2}, 2^{-3}$	$-2^{-2}, -2^{-3}$

At each P branch of Fig. 3-2, the T and, hence, V values are known. All the possible gradient values are generated from the possible POTTs available for selection. Every possible gradient value is considered as a potential solution for each new region derived from dividing the f-regions and error analysis is performed to determine its validity as described below.

3.7 Error Analysis

Error analysis is performed for each and every possible gradient value for each new region derived from dividing the f-regions. Invalid gradient values that violate the accuracy requirement are rejected. Let (3.13) be the approximation function of a region $g_r = [L_r, U_r]$.

$$LA_r(1+m) = m + C_r \cdot m_{MSB}^* + D_r \quad (3.13)$$

For efficient computation, it is proposed in this research work to perform accuracy analysis on only the small values of N , $1 \leq N \leq 2^{10} - 1$, even though the input wordlength of N is 32 bits, as the *MAEP*, which defines the accuracy, usually associates with a small N value ($< 2^8$ in all the existing Mitchell-based methods). The error analysis for the gradient C_r in the region g_r is performed on the range of N that falls within the same region g_r .

For the regions g_r with lower bound $L_r \neq 1$ and upper bound $U_r \neq 2$, i.e. other than the first and last region, the accuracy check is performed by computing (3.15) and (3.16) to determine, respectively, the values $D_{r.maxl}$ and $D_{r.minu}$ for the constant D_r in (3.13) within the region. If the maximum value of the D_r lower bound ($D_{r.maxl}$) is less than or equal to the minimum value of the D_r upper bound ($D_{r.minu}$), i.e., $D_{r.maxl} \leq D_{r.minu}$, there exists a constant D_r such that (3.14) holds for all N , i.e. each and every *EP* (error percentage) is less than *Acc* and the accuracy requirement is satisfied. Hence, the corresponding gradient value C_r is accepted. Otherwise, the gradient value is rejected.

$$|EP| = \left| \frac{(\log_2(1+m) - LA_r(1+m))}{\log_2(N)} \cdot 100 \right| \leq Acc \quad (3.14)$$

$$\Rightarrow -\frac{Acc}{100} \leq \frac{(\log_2(1+m) - (m + C_r \cdot m_{MSB}^* + D_r))}{\log_2(N)} \leq \frac{Acc}{100}$$

$$D_{r.l} = \log_2(1 + m) - m - C_r \cdot m_{MSB}^* - \frac{Acc}{100} \cdot \log_2(N)$$

$$D_{r.maxl} = \max_{\forall N} D_{r.l} \quad (3.15)$$

$$D_{r.u} = \log_2(1 + m) - m - C_r \cdot m_{MSB}^* + \frac{Acc}{100} \cdot \log_2(N)$$

$$D_{r.minu} = \min_{\forall N} D_{r.u} \quad (3.16)$$

For the first and last region, of which $L_r = 1$ and/or $U_r = 2$, $D_r = 0$ is used in (3.13) for accuracy check as setting $D_r = 0$ leads to better solutions in these two regions. Furthermore, an additional check is performed to ensure no overflow in (3.13). If carry is generated in (3.13), it will propagate to the characteristic k and k has to be included in the approximation circuit, resulting in more complex hardware. There is no overflow if $0 \leq LA_r(1 + m) < 2$ for all N that falls within these two regions and a C_r value is accepted only if it does not cause overflow.

After invalid gradient values are rejected, there may be still more than one valid gradient values for a region. In such cases, only those with the fewest non-zero POTTs are kept while the rest are discarded. Still, there may be more than one valid gradients for each region and, hence, more than one solutions available like the example of (3.12). The best solution is determined as described below.

3.8 Selection of Gradient

Despite only the valid combinations with the fewest POTTs are kept, there may be still more than one solutions. Among them, only the one with the least hardware complexity is chosen. The selection of the best solution, i.e. the best combinations of POTTs for the gradients of the regions, is explained below with the valid POTTs listed in Table 3-6 as an illustrative example.

In the example, the number of regions is $R = 5$ and maximum number of POTTs is $P = 3$. Under each column, each C_r , containing at most $P = 3$ POTTs ($c_{r,1}$, $c_{r,2}$ and $c_{r,3}$), is a valid gradient for the region g_r . For instance, there are four valid C_2 , one of which is $C_2 = 2^{-1} + 2^{-2} + 2^{-6}$, whereas there is only one C_3 , which is $C_3 = 2^{-2} + 2^{-5} + 2^{-6}$. A solution can be obtained by

TABLE 3-6: AN EXAMPLE OF C_r

C_1	C_2	C_3	C_4	C_5
$c_{1,1}, c_{1,2}, c_{1,3}$	$c_{2,1}, c_{2,2}, c_{2,3}$	$2^{-2}, 2^{-5}, 2^{-6}$	$c_{4,1}, c_{4,2}, c_{4,3}$	$c_{5,1}, c_{5,2}, c_{5,3}$
$c_{1,1}, c_{1,2}, c_{1,3}$	$2^{-1}, 2^{-2}, 2^{-6}$		$c_{4,1}, c_{4,2}, c_{4,3}$	$c_{5,1}, c_{5,2}, c_{5,3}$
$2^{-5}, 0, 0$	$c_{2,1}, c_{2,2}, c_{2,3}$		$2^{-2}, 2^{-6}, 2^{-7}$	$c_{5,1}, c_{5,2}, c_{5,3}$
$c_{1,1}, c_{1,2}, c_{1,3}$	$c_{2,1}, c_{2,2}, c_{2,3}$			$c_{5,1}, c_{5,2}, c_{5,3}$
$c_{1,1}, c_{1,2}, c_{1,3}$				$2^{-2}, 2^{-5}, 2^{-6}$
$c_{1,1}, c_{1,2}, c_{1,3}$				

selecting one C_r from each column. As such, there are $6 \times 4 \times 1 \times 3 \times 5 = 360$ solutions in this example.

The hardware merits, such as area cost, critical path delay and area-delay product (ADP), of each of the solutions are first evaluated as explained below with the approximation function (3.18) as an example, of which the gradients are the POTTs with explicit values from Table 3-6.

$$\begin{aligned} & LA_{5,f}(1+m) \\ &= \begin{cases} m + (0 + 0 + 2^{-5}) \cdot m_{MSB}^* + 0 \\ m + (2^{-1} + 2^{-2} + 2^{-6}) \cdot m_{MSB}^* + D_2 \\ m + (2^{-2} + 2^{-5} + 2^{-6}) \cdot m_{MSB}^* + D_3 \\ m + (2^{-2} + 2^{-6} + 2^{-7}) \cdot m_{MSB}^* + D_4 \\ m + (2^{-2} + 2^{-5} + 2^{-6}) \cdot m_{MSB}^* + 0 \end{cases} \end{aligned} \quad (3.17)$$

$$\begin{aligned} &= \begin{cases} m + (0 + 0 + 2^{-5}) \cdot m_{MSB}^* + 0 \\ m + (2^{-1} + 2^{-2} + 2^{-6}) \cdot m_{MSB}^* + D_2 \\ m + (2^{-6} + 2^{-2} + 2^{-5}) \cdot m_{MSB}^* + D_3 \\ m + (2^{-6} + 2^{-2} + 2^{-7}) \cdot m_{MSB}^* + D_4 \\ m + (2^{-6} + 2^{-2} + 2^{-5}) \cdot m_{MSB}^* + 0 \end{cases} \end{aligned} \quad (3.18)$$

where $D_{r.maxl} \leq D_r \leq D_{r.minu}$, ($r = \{2,3,4\}$), is a constant value truncated at 2^{-T} binary position ($T = 8$ is assumed in this example). Based on the generic approximation architecture in Fig. 3-1, the circuit for (3.18) is given in Fig. 3-4(a) with the details of its adder tree in Fig. 3-5. The hardware merits of Fig. 3-4(a) are computed based on the unit-gate model described in Section 3.3.

The total area cost is the sum of the area cost of the adder tree and MUXes. Given the operands from (3.18), the number of FAs and HAs required in the adder tree is computed based on Dadda's adder tree architecture and the area cost and critical path delay are then calculated accordingly. As shown in Fig. 3-5, the adder tree requires 22 FAs and 7 HAs, leading to 175 units of area as shown in Table 3-7. The number of 2-to-1 MUXes needed to obtain I_1 , I_2 , I_3 and D are 14, 6, 6 and 24 respectively, incurring a total area of 100 units. In total, the area cost of the approximation circuit in Fig. 3-4(a) is 275.

The total critical path delay is the sum of the delay of the critical paths in the adder tree and multi-input MUXes. The adder tree in Fig. 3-5 has three reduction stages and an 8-bit RCA, leading to a delay of 42 units. For the multi-input MUXes, the critical path lies in the one with the most inputs. For the circuit in Fig. 3-4(a), the critical path lies in MUX_D while the delays of the other MUXes are immaterial. Implemented with 2-to-1 MUXes, 4-input MUX has $\lceil \log_2(4) \rceil = 2$ 2-to-1 MUXes

Fig. 3-4: Approximation circuit for: (a) (3.18), (b) (3.17)

Fig. 3-5: Adder tree for approximation circuits in Fig. 3-4

in the critical path. In total, the critical path delay of the circuit in Fig. 3-4(a) is 46 units. Therefore, $ADP = 275 \times 46 = 12650$.

It is worth noting that (3.17) has the same gradients as (3.18) but some of the POTTs are at different positions. In other words, (3.18) is obtained from (3.17) by re-arranging the positions of the POTTs in (3.17), i.e., the same POTT values are aligned at the same columns. With the rearrangement of POTTs, a more optimized circuit is obtained. Should the approximation function (3.17) be implemented, the circuit would have been the one in Fig. 3-4(b) (the adder tree remains the same as in Fig. 3-5) and the area cost, critical path delay and ADP would have been 299, 46

Fig. 3-6: MUX for operand I_1 : (a) without optimization, (b) with optimization

TABLE 3-7: AREA AND DELAY FOR CIRCUIT IN FIG. 3-4

Circuits		Fig. 3-4(a)		Fig. 3-4(b)	
		A	τ	A	τ
2-to-1 MUX	I_1	$14 \times 2 = 28$	-	$14 \times 2 = 28$	-
	I_2	$6 \times 2 = 12$	-	$18 \times 2 = 36$	-
	I_3	$6 \times 2 = 12$	-	$6 \times 2 = 12$	-
	D	$24 \times 2 = 48$	$2 \times 2 = 4$	$24 \times 2 = 48$	$2 \times 2 = 4$
HA		$7 \times 3 = 21$	$1 \times 2 = 2$	$7 \times 3 = 21$	$1 \times 2 = 2$
FA		$22 \times 7 = 154$	$10 \times 4 = 40$	$22 \times 7 = 154$	$10 \times 4 = 40$
Total		275	46	299	46

and 13754 units respectively as shown in Table 3-7. The circuit of Fig. 3-4(a) has better area by 8.0%. For fast computation, a greedy fixed-bin packing method is used in the proposed UMA to determine the positions of the POTTs.

In the method, the POTTs are first sorted within each region in a sequence starting with zeros followed by POTTs in the descending order of their magnitudes. Given a P value, there are P bins for packing the POTTs such that not more than one POTT from the same region is packed into the same bin and the same POTT from different regions is packed into the same bin as many as possible. The 2-phase greedy method starts the packing by placing the POTTs in the first region into one bin each. The POTTs of the subsequent regions g_r , $r = 2$ to $r = R$ ($R > 1$), are then packed term by term and region by region. When packing a POTT of a region, it is placed in the first bin that does not contain another POTT of the same region but has at least one matched POTT from another region. If no such bin is available for the POTT, it is skipped and packed in the second phase into any bin that does not contain a POTT from the same region.

After the rearrangement of POTTs, further optimization is performed for the regions $r = 2$ to $r = R - 1$ if $R > 2$ for the special cases where some of the POTTs in the gradient values are 0 as explained below with (3.19), modified from (3.18), as an illustrative example. Based on the generic circuit in Fig. 3-1, the circuit for (3.19) would include a 5-operand adder tree and the MUXes for operands I_1 and D shown in Fig. 3-6(a).

$$\begin{aligned}
& \text{LA}_{5,g}(1+m) \\
&= \begin{cases} m + (0 + 0 + 2^{-5}) \cdot m_{MSB}^* + 0 \\ m + (0 + 2^{-2} + 2^{-6}) \cdot m_{MSB}^* + D_2 \\ m + (0 + 2^{-2} + 2^{-5}) \cdot m_{MSB}^* + D_3 \\ m + (0 + 2^{-2} + 2^{-7}) \cdot m_{MSB}^* + D_4 \\ m + (2^{-6} + 2^{-2} + 2^{-5}) \cdot m_{MSB}^* + 0 \end{cases} \quad (3.19)
\end{aligned}$$

$$\begin{aligned}
&= \begin{cases} m + (0 + 0 + 2^{-5}) \cdot m_{MSB}^* \\ m + (D_2 + 2^{-2} + 2^{-6}) \cdot m_{MSB}^* \\ m + (D_3 + 2^{-2} + 2^{-5}) \cdot m_{MSB}^* \\ m + (D_4 + 2^{-2} + 2^{-7}) \cdot m_{MSB}^* \\ m + (2^{-6} + 2^{-2} + 2^{-5}) \cdot m_{MSB}^* \end{cases} \quad (3.20)
\end{aligned}$$

In such a special case, the MUXes for operands I_1 and D in Fig. 3-6(a) are merged into a single MUX as shown in Fig. 3-6(b). Also, it can be seen from (3.20) that the number of operands is now reduced to only four, and hence reducing the number of FAs and HAs in the adder tree. In general, when the POTTs of any region $r = 2$ to $R - 1$ ($R > 2$) are 0, the constant term of the same region will take their place to further optimize the circuit.

After the hardware merits of all the valid gradients are evaluated, the best design in term of area cost, critical path delay or area-delay product is determined and the corresponding hardware merit is recorded.

3.9 Algorithm

With the proposed Unified Mitchell Approximation method described in Sections 3.2.1 – 3.2.7, the algorithm for the design space exploration is formulated with its pseudo code shown in Fig. 3-7, where “Acc” is the accuracy requirement. The “for” loop sweeps the parameter T and the two “repeat” loops, Loop_P and Loop_R, correspond to the search at the P and R levels of the design space tree in Fig. 3-2. The function **findAllGradients**(P, V) finds all the possible gradients described in Section 3.6. The function **analyzeErr**(C, g_r) checks the validness of the gradients as described in Section 3.7 and returns a set of valid gradients (in C_r) for the region g_r and a ‘*valid_r*’ flag indicating whether g_r is a f-region (*valid_r* = 0). The f-regions are divided by the function **divideFailRegion**(g_r) until all become s-regions as described in Section 3.5. When a solution is found (at least one valid gradient value is found for each region, i.e., Bound2 of Fig. 3-2 is reached), Loop_R is terminated and the function **determineBestDesign**($\forall C_r$) selects the best combination of gradients (best hardware complexity) as described in Section 3.8. The design is the best one under a “P” branch of Fig. 3-2 and recorded as the *PBranchBest*. Note that the example

```

UMA(Acc)
{
   $T_0 = \lfloor -\log_2(Acc) \rfloor$ ;
  bestOfAll =  $\phi$ ;
  for  $T = T_0 - 3$  to  $T_0 + 4$  do           // sweep  $T$ 
  {
     $P = 0$ ;
     $V = 2^{-(T-1)}$ ;
    TBranchBest =  $\phi$ ;
  Loop_P: repeat
  {
     $R = 1$ ;
     $r = 1$ ;
     $g_1 = [1, 2]$ ;
     $C = \text{findAllGradients}(P, V)$ ;
  Loop_R: repeat
  {
    [validr,  $C_r$ ] = analyzeErr ( $C$ ,  $g_r$ );
    if validr = 0 do
    {
      divideFailRegion ( $g_r$ );
       $R = R + 1$ ;
    }
    else  $r = r + 1$ ;
  } until  $r > R$ ;           // end Loop_R
  PBranchBest = determineBestDesign ( $\forall C_r$ );
  [TBranchBest, Improve] = compareDesigns
    (TBranchBest, PBranchBest);
   $P = P + 1$ ;
  } until (Improve = 0 or  $P = T$ ); // end Loop_P
  [bestOfAll, Improve] = compareDesigns (bestOfAll,
    TBranchBest);
  }                               // end "for  $T$ " loop
   $B = -\log_2(\text{minRegionSize}(\text{bestOfAll}))$ ;
}

```

Fig. 3-7: Pseudo code of proposed UMA

of (3.18) described in Section 3.8 is the *PBranchBest* for the branch $P = 3$. The *TBranchBest* is the best design under a “ T ” branch of Fig. 3-2. The function **compareDesigns**(*oldDesign*, *newDesign*) returns the better design of the two and a flag indicating an improvement if *newDesign* is the better one. If there is no improvement or P reaches its limit ($T - 1$), “Bound1” of Fig. 3-2 is reached and Loop_P is terminated. The search continues with the next T and the final best design, *bestOfAll*, is the best design of all the *TBranchBest* designs. The function **minRegionSize**(*aDesign*) finds the size of the smallest region in *aDesign*.

3.10 Performance Evaluation and Discussions

3.10.1 Comparisons with Mitchell-based Methods

The proposed search algorithm is coded in Matlab and executed to obtain an optimized design (approximation function) for each of the accuracies (in term of *MAEP*) achieved by the existing Mitchell-based (MA) methods as shown in Table 3-8. For each existing design, the first row is the *MAEP* achieved by the design and it is used as “*Acc*” in UMA. The *MAEP*, area, delay and *ADP* achieved by UMA are shown in the lower part of the table. For example, for the *MAEP* = 0.189% achieved by De Caro et al. in their 4-region approximation [DeC11], UMA obtains a 15–

TABLE 3-8: ACCURACY AND COMPLEXITY COMPARISONS AMONG THE DESIGNS OF THE PROPOSED METHOD AND THE EXISTING MITCHELL-BASED METHODS

Existing MA Methods		Mitchell [Mit62] 1962	Combet [Com65] 1965	Hall [Hal70] 1970	San-Gregory [San99] 1999	Abed [Abe03] 2003			Kim [Kim06] 2006	Li [Li08] 2008	Juang [Jua09] 2009	Gutierrez [Gut11] 2011	De Caro [DeC11] 2011		
MAEP (%)		5.361	0.267	0.781	1.540	0.930	0.431	0.154	0.050	4.304	1.079	0.793	0.582	0.189	0.032
A (μm) ²		4553	11133	17771	5602	5135	5836	6358	13816	5747	6342	8551	6832	8311	9807
τ (ns)		0.67	4.14	3.95	1.02	0.93	0.99	1.30	3.47	1.51	1.51	1.87	1.86	1.86	2.49
ADP		3051	46091	70195	5714	4776	5778	8265	47942	8678	9576	15990	12708	15458	24419
The proposed UMA	LA _{1,K} (1+m)	LA _{4,F} (1+m)	LA _{4,C} (1+m)	LA _{2,B} (1+m)	LA _{2,B} (1+m)	LA _{4,E} (1+m)	LA _{6,H} (1+m)	LA _{11,I} (1+m)	LA _{2,A} (1+m)	LA _{2,B} (1+m)	LA _{4,C} (1+m)	LA _{3,D} (1+m)	LA _{15,G} (1+m)	LA _{10,J} (1+m)	
	Time (s)	<1	122	7	4	5	17	46	85	2	5	7	7	175	824
MAEP (%)		5.361	0.267	0.693	0.930	0.930	0.431	0.147	0.048	3.098	0.930	0.693	0.554	0.167	0.032
A (μm) ²		4553	6257	5998	5135	5135	5836	5545	8276	5532	5135	5998	5675	5862	8926
τ (ns)		0.67	1.37	0.95	0.93	0.93	0.99	1.43	1.69	0.71	0.93	0.95	1.03	1.13	1.90
ADP		3051	8572	5698	4776	4776	5778	7929	13986	3928	4776	5698	5845	6624	16959
Δ (%)	MAEP	0	0	11.3	39.6	0	0	4.5	4.0	28.0	13.8	12.6	4.8	11.6	0
	A	0	43.8	66.2	8.3	0	0	12.8	40.1	3.7	19.0	30.0	16.9	29.5	9.0
	τ	0	66.9	75.9	8.8	0	0	-10.0	51.3	53.0	38.4	49.2	44.6	39.2	23.7
	ADP	0	81.4	91.9	16.4	0	0	4.1	70.8	54.7	50.1	64.4	54.0	57.1	30.5

region design, $LA_{15,G}(1+m)$ in 175 seconds with a better accuracy of 0.167%, yielding an improvement of 11.6% in *MAEP*, as well as an improvement in area, delay and *ADP* by 29.5%, 39.2% and 57.1% respectively, despite more regions are used. The detailed designs (approximation functions in the form of (3.3)) obtained by UMA are given in Table 3-9. For example, the 3-region design, $LA_{3,D}(1+m)$, with $MAEP = 0.554\%$ has the gradients and constants as given in Table 3-9 and can be expressed as:

$$\begin{aligned}
 & LA_{3,D}(1+m) \\
 &= \begin{cases} m + 2^{-2} \cdot m_{MSB}^* + 0 & m \in [0,0.5) \\ m & m \in [0.5,0.75) \\ m - 2^{-2} \cdot m_{MSB}^* + 0 & m \in [0.75,1) \end{cases} \\
 &= \begin{cases} m + 2^{-2} \cdot m_{MSB4} + 0 & m \in [0,0.5) \\ m & m \in [0.5,0.75) \\ m + 2^{-2} \cdot \bar{m}_{MSB4} + 0 & m \in [0.75,1) \end{cases}
 \end{aligned}$$

The designs achieved by the existing MA methods as well as those by the proposed UMA are modeled and simulated in Matlab with 32-bit integer input N and error percentage (EP) computed based on (3.21). The double-precision logarithm function in Matlab is used to emulate the accurate values of $\log_2(N)$. The *MAEP* results obtained from the simulations are included in Table 3-8.

$$EP = \frac{(\log_2(1+m) - LA_{R,\#}(1+m))}{\log_2(N)} \times 100. \quad (3.21)$$

TABLE 3-9: DESIGNS OBTAINED BY THE PROPOSED SEARCH ALGORITHM FOR ACCURACIES ACHIEVED BY THE EXISTING DESIGNS

Designs	T	g_r	C_r	$D_r (\times 2^{-T})$	Designs	T	g_r	C_r	$D_r (\times 2^{-T})$
$LA_{2,A}(1+m)$ ($MAEP = 3.098\%$)	3	[1.0,1.5)	0	0	$LA_{6,H}(1+m)$ ($MAEP = 0.147\%$)	8	[1.0000,1.0625)	2^{-2}	0
		[1.5,2.0)	2^{-1}	0			[1.0625,1.1250)	2^{-2}	3
$LA_{2,B}(1+m)$ ($MAEP = 0.930\%$)	5	[1.0,1.5)	2^{-2}	0			[1.1250,1.2500)	2^{-2}	3
		[1.5,2.0)	-2^{-2}	0			[1.2500,1.5000)	2^{-4}	15
$LA_{4,C}(1+m)$ ($MAEP = 0.693\%$)	5	[1.000,1.500)	2^{-2}	0			[1.5000,1.7500)	-2^{-3}	7
		[1.500,1.750)	0	3			[1.7500,2.0000)	-2^{-2}	0
		[1.750,1.875)	0	2	$LA_{11,I}(1+m)$ ($MAEP = 0.048\%$)	10	[1.00000,1.03125)	2^{-1}	0
		[1.875,2.000)	-2^{-1}	0			[1.03125,1.06250)	2^{-1}	-2
$LA_{3,D}(1+m)$ ($MAEP = 0.554\%$)	6	[1.00,1.50)	2^{-2}	0			[1.06250,1.12500)	2^{-2}	11
		[1.50,1.75)	0	6			[1.12500,1.25000)	2^{-2}	14
		[1.75,2.00)	-2^{-2}	0			[1.25000,1.37500)	2^{-3}	42
$LA_{4,E}(1+m)$ ($MAEP = 0.431\%$)	6	[1.00,1.25)	2^{-2}	0			[1.37500,1.50000)	0	86
		[1.25,1.50)	0	5			[1.50000,1.62500)	-2^{-4}	56
		[1.50,1.75)	0	5			[1.62500,1.75000)	-2^{-3}	30
		[1.75,2.00)	-2^{-2}	0			[1.75000,1.81250)	-2^{-2}	-4
$LA_{4,F}(1+m)$ ($MAEP = 0.267\%$)	7	[1.00,1.25)	$2^{-2} + 2^{-3}$	0			[1.81250,1.87500)	-2^{-2}	-2
		[1.25,1.50)	0	10			[1.87500,2.00000)	-2^{-2}	0
		[1.50,1.75)	-2^{-3}	4	$LA_{10,J}(1+m)$ ($MAEP = 0.032\%$)	10	[1.0000,1.0625)	$2^{-2} + 2^{-3}$	0
$LA_{15,G}(1+m)$ ($MAEP = 0.167\%$)	7	[1.75,2.00)	-2^{-2}	0			[1.0625,1.1250)	$2^{-2} + 2^{-3}$	1
		[1.00000,1.03125)	0	0			[1.1250,1.2500)	$2^{-2} - 2^{-5}$	-12
		[1.03125,1.06250)	0	2			[1.2500,1.3750)	2^{-3}	41
		[1.06250,1.12500)	0	4			[1.3750,1.5000)	0	87
		[1.12500,1.18750)	0	6			[1.5000,1.6250)	-2^{-4}	56
		[1.18750,1.25000)	0	8			[1.6250,1.7500)	-2^{-3}	30
		[1.25000,1.31250)	0	9			[1.7500,1.8750)	$-2^{-3} - 2^{-4}$	12
		[1.31250,1.37500)	0	10			[1.8750,1.9375)	-2^{-2}	2
		[1.37500,1.50000)	0	11			[1.9375,2.0000)	$-2^{-2} - 2^{-4}$	0
		[1.50000,1.62500)	0	11					
		[1.62500,1.75000)	0	9					
		[1.75000,1.81250)	0	7					
		[1.81250,1.87500)	0	6					
		[1.87500,1.93750)	0	4					
		[1.93750,1.96875)	0	2					
		[1.96875,2.00000)	0	0					

The designs obtained by UMA are simulated again even though they had passed the accuracy check by the function **analyzeErr** (Section 3.7) because only 10-bit integer input N is considered in the function **analyzeErr** for fast computation. As such, the accuracy might be violated when the wordlength of N increases. If the accuracy is not met in the 32-bit simulations, UMA is re-executed for the same accuracy with larger wordlength of N in the function **analyzeErr**. Specifically, the function **analyzeErr** will now perform error checking for $1 \leq N \leq 2^{n_e} - 1$, where $n_e = \lceil \log_2(N_{maxerr}) \rceil$ and N_{maxerr} is N at which the maximum error occurs, so that UMA is able to find a new approximation function without violating the accuracy requirement at

N_{maxerr} . Note that violations might occur at other values of N for the new approximation function and the full wordlength simulation is always performed after each approximation function obtained. Such a situation occurs for the accuracy requirement of 2^{-14} (14 bits) when UMA is re-executed with $n_e = 13$ ($N_{maxerr} = 4479$). The approximation function for the accuracy requirement of 14 bits is presented in the next section.

Based on the structure of Fig. 2-1 with the architecture of the approximation circuit in Fig. 3-1, the designs achieved by the existing MA methods and UMA are coded in Verilog HDL for functional verification and synthesized in STM 90 nm CMOS technology using Synopsys Design Compiler. The synthesis results, area (A) and delay (τ), and the corresponding area-delay product (ADP) computed are also included in Table 3-8. Note that UMA uses ADP as the optimization goal. Except for the original design by Mitchell [Mit62] and the 2- and 3-region designs by Abed and Siferd [Abe03], the designs obtained by UMA are better than the corresponding existing Mitchell-based methods. It is worth noting that eight of the thirteen designs achieve improvement (Δ) by more than 50% in ADP as compared to the corresponding designs by the existing Mitchell-based methods. When the $MAEP$ of the Mitchell's design is given as the accuracy requirement to UMA, the same design, i.e. without approximation circuit, is obtained. For the $MAEP$ of the 2- and 3-region designs by Abed and Siferd, UMA obtains exactly the same designs. Note that the two regions g_2 and g_3 of $LA_{4,E}(1 + m)$ have the same equation, making it effectively the same as $LA_{3,Abed}(1 + m)$ by Abed and Siferd.

The time (Time) needed to execute the proposed search algorithm on a workstation with Intel Core2 Quad CPU (2.66GHz, 1.98GB RAM) is also provided in Table 3-8. Among the designs achieved, the longest time needed is 13 minutes (824 seconds) for the accuracy requirement of $MAEP = 0.032\%$ achieved by De Caro et al.

3.10.2 Scalability of UMA

To investigate the scalability and the correlations between the estimated and synthesis results, UMA is also executed to obtain designs with the accuracy increased bit by bit starting from 4 bits. Table 3-10 lists the time needed to obtain the designs, the estimated and the synthesis results. The computation time increases significantly for accuracy higher than 12 bits. For accuracy of 15 bits, the search took too long and it was abandoned. Together with the comparison results in Table 3-8, it is evident that the proposed UMA is scalable and able to obtain an optimized logarithmic converter for any desired accuracy up to 14 bits within practical period of time.

TABLE 3-10: ESTIMATED AND SYNTHESIS RESULTS OF THE HARDWARE MERITS

Accuracy	Time (s)	Estimation			Synthesis		
		A	τ	ADP	A (μm^2)	τ (ns)	ADP
4 bits	<1	548	28	15344	4553	0.67	3051
5 bits	3	565	38	21470	5532	0.71	3928
6 bits	4	577	44	25388	5135	0.93	4776
7 bits	7	609	50	30450	5998	0.95	5698
8 bits	20	664	64	42496	5679	1.30	7383
9 bits	169	716	66	47256	7026	1.36	9555
10 bits	70	876	66	57816	7268	1.46	10611
11 bits	112	960	78	74880	8276	1.69	13986
12 bits	1525	1091	84	91644	9944	2.08	20684
13 bits	7759	1363	86	117218	11895	1.97	23433
14 bits	49038	1606	90	144540	13410	2.17	29100

(a) Estimated ADP

(b) Synthesis ADP

(c) Normalized ADP

Fig. 3-8: ADP of the approximation functions

The designs obtained for accuracies from 4 bits to 14 bits are given in Table 3-11. They are modeled and simulated in Matlab to verify the accuracy. For the accuracy of 14 bits, UMA took more than 13 hours to obtain the optimized design. For the accuracy of 15 bits, the execution was abolished after two weeks and no optimized design was obtained. Hence, the current version of UMA is able to obtain any designs with accuracy up to 14 bits. The designs obtained are modeled in Verilog HDL for functional verification and synthesized in STM 90 nm CMOS technology by

Synopsys Design Compiler. The estimated and synthesis results for *ADP* are included in Table 3-10 and plotted in Fig. 3-8 (a) and (b) respectively. Both the results are then normalized to their maximum values and plotted in Fig. 3-8 (c). It is evident that the two curves in Fig. 3-8 (c) have well-matched trends, demonstrating the efficacy of the hardware model and the estimation method.

TABLE 3-11: DESIGNS OBTAINED BY THE PROPOSED SEARCH ALGORITHM FOR ACCURACIES OF 4 BITS TO 14 BITS

Acc (bits)	Designs	T	g_r	C_r	D_r ($\times 2^{-T}$)	Acc (bits)	Designs	T	g_r	C_r	D_r ($\times 2^{-T}$)
4	No approximation is needed					13	$LA_{19,0}(1+m)$ (MAEP = 0.0118%)	11	[1.000000,1.015625)	2^{-1}	0
5	Same as $LA_{2A}(1+m)$ in Table 3-9								[1.015625,1.031250)	2^{-1}	-3
6	Same as $LA_{2B}(1+m)$ in Table 3-9								[1.031250,1.062500)	$2^{-2} + 2^{-3}$	4
7	Same as $LA_{4C}(1+m)$ in Table 3-9								[1.062500,1.125000)	$2^{-2} + 2^{-4}$	12
8	$LA_{5,K}(1+m)$ (MAEP = 0.3359%)	8	[1.000,1.125)	2^{-1}	0				[1.125000,1.187500)	2^{-2}	28
			[1.125,1.250)	0	13				[1.187500,1.250000)	$2^{-2} - 2^{-4}$	-75
			[1.250,1.500)	0	20				[1.250000,1.312500)	2^{-3}	83
			[1.500,1.750)	0	21				[1.312500,1.375000)	2^{-4}	124
			[1.750,2.000)	-2^{-2}	0				[1.375000,1.437500)	2^{-5}	149
9	$LA_{4,L}(1+m)$ (MAEP = 0.1849%)	8	[1.00,1.25)	$2^{-2} + 2^{-4}$	0	[1.437500,1.500000)	0	176			
			[1.25,1.50)	2^{-4}	15	[1.500000,1.562500)	-2^{-4}	111			
			[1.50,1.75)	-2^{-3}	7	[1.562500,1.625000)	$-2^{-4} - 2^{-5}$	84			
			[1.75,2.00)	-2^{-2}	0	[1.625000,1.687500)	-2^{-3}	60			
10	$LA_{15,M}(1+m)$ (MAEP = 0.0971%)	7	[1.00000,1.06250)	2^{-1}	0	[1.687500,1.750000)	$-2^{-3} - 2^{-5}$	40			
			[1.06250,1.09375)	2^{-1}	-1	[1.750000,1.812500)	$2^{-4} - 2^{-2}$	-106			
			[1.09375,1.12500)	0	5	[1.812500,1.875000)	$2^{-5} - 2^{-2}$	-54			
			[1.12500,1.25000)	2^{-2}	2	[1.875000,1.937500)	-2^{-2}	2			
			[1.25000,1.31250)	2^{-2}	1	[1.937500,1.968750)	-2^{-2}	2			
			[1.31250,1.37500)	0	10	[1.968750,2.000000)	$-2^{-2} - 2^{-4}$	0			
			[1.37500,1.50000)	0	11	[1.0000000,1.0078125)	2^{-1}	0			
			[1.50000,1.62500)	-2^{-3}	4	[1.0078125,1.0156250)	2^{-1}	-2			
			[1.62500,1.68750)	-2^{-2}	-1	[1.0156250,1.0312500)	$2^{-1} - 2^{-3}$	-506			
			[1.68750,1.75000)	-2^{-2}	0	[1.0312500,1.0625000)	$2^{-1} - 2^{-3}$	-504			
			[1.75000,1.81250)	0	7	[1.0625000,1.0781250)	$2^{-1} - 2^{-3}$	-505			
			[1.81250,1.87500)	-2^{-2}	1	[1.0781250,1.0937500)	$2^{-1} - 2^{-3}$	-508			
[1.87500,1.93750)	-2^{-2}	1	[1.0937500,1.1250000)	$2^{-2} + 2^{-4}$	26						
11	Same as $LA_{11,I}(1+m)$ in Table 3-9					14	$LA_{23,P}(1+m)$ (MAEP = 0.0061%)	12	[1.1250000,1.1562500)	$2^{-2} + 2^{-5}$	40
12	$LA_{11,N}(1+m)$ (MAEP = 0.0243%)	11	[1.96875,2.00000)	-2^{-1}	0				[1.1562500,1.1875000)	2^{-2}	58
									[1.1875000,1.2500000)	$2^{-2} - 2^{-4}$	-151
									[1.2500000,1.3125000)	$2^{-3} + 2^{-7}$	159
									[1.3125000,1.3750000)	$2^{-4} + 2^{-6}$	228
									[1.3750000,1.4375000)	2^{-5}	298
									[1.4375000,1.5000000)	0	318
									[1.5000000,1.5625000)	$2^{-6} - 2^{-4}$	189
									[1.5625000,1.6250000)	$2^{-5} - 2^{-3}$	38
									[1.6250000,1.6875000)	-2^{-3}	118
									[1.6875000,1.7500000)	$-2^{-3} - 2^{-5}$	78
									[1.7500000,1.8125000)	$2^{-4} - 2^{-2}$	-202
									[1.8125000,1.8750000)	$2^{-5} - 2^{-2}$	-108
			[1.8750000,1.9375000)	-2^{-2}	4						
			[1.9375000,1.9687500)	-2^{-2}	6						
			[1.9687500,2.0000000)	$-2^{-2} - 2^{-5}$	0						

3.11 Summary

The Mitchell-based approximation approach for computation of binary logarithm in integrated circuits is studied. The design parameters involved in the Mitchell-based method and their impact on accuracy and hardware merits are analyzed. The hardware model of the approximation circuit

for performance merit evaluation is formulated. Consequently, a novel method named the Unified Mitchell-based Approximation (UMA) is proposed to obtain an optimized circuit for any desired logarithmic conversion accuracy. The proposed method explores a design space consisting of five design parameters that impact the accuracy and hardware merits. The design space of the five parameters is too huge for exhaustive search. Constraints are proposed for the parameters and a new search algorithm is proposed to sweep across the range of the parameter values so that the search can be completed within practical period of time without compromising the optimization objective. UMA is executed to obtain an optimized approximation function for each of the accuracies achieved by all the existing Mitchell-based methods. Synthesis results show that the designs obtained by UMA are the same as or better than the corresponding existing Mitchell-based designs. Eight of the thirteen designs obtained achieve improvement by more than 50% in area-delay product. UMA is scalable and able to find an optimized circuit for any accuracy requirement up to 14 bits.

Despite the designs obtained by UMA being optimized and having better hardware merits than the existing designs, they are not necessary the best design. The design space is not exhaustively explored due to its huge size. In particular, the divisions of regions for approximation do not consider all possible combinations. On the other hand, the constraints on the five design parameters need to be further tighten so that the design space exploration is more practical for accuracy higher than 14 bits. To speed up the search for higher accuracy, heuristic approaches may be considered, which will trade-off between computation time and hardware merits. Alternatively, the proposed UMA can work together with the range mapping technique proposed in the next chapter to achieve higher accuracy.

CHAPTER 4. Range Mapping – A Fresh Approach to High Accuracy Mitchell-based Logarithmic Conversion

4.1 Overview

Approaching the Mitchell-based logarithmic conversion from a fresh angle, a novel technique named the range mapping method is proposed. The range mapping method compresses the Mitchell range, $M = [1, 2)$, into a new range, $[1.25, 1.5625)$, that is less than one third of its original size. Not only is the compressed range smaller, the binary logarithmic function to be approximated is more linear. These advantageous properties enhance the effectiveness of the piecewise linear approximation in approximating the logarithmic function. Hence, when the conventional Mitchell-based piecewise linear approximation (Section 2.1) is applied to approximate the logarithmic function within the compressed range, significant improvement in the logarithmic conversion accuracy is achieved without aggravating the circuit complexity. The following sections describe in details the mathematical formulations of the proposed method.

4.2 Range Mapping (RM)

The general idea of range mapping is to divide the Mitchell range, M , into several segments. Each segment is then scaled and mapped individually such that all the segments fall within a range smaller than M . The compressed range is then approximated in the same way as the Mitchell's method and the result are scaled back. Mathematically, the mapping for each segment i is based on the algebraic relation (4.1), where G_i is a constant to scale the segment and $1 + m_2$ is the mapped range. The challenge is to find a set of valid and feasible constants G_i such that the constants contain only, and as few as possible, power-of-two terms (POTTs) for efficient circuit implementation. In addition, the compressed range is small to achieve high accuracy by using piecewise linear approximation. The formulation and the mapping method are described as follows.

$$\begin{aligned}
 \log_2 N &= k + \log_2(1 + m) \\
 &= k + \log_2[G_i(1 + m)] - \log_2 G_i \\
 &= k + \log_2(1 + m_2) - \log_2 G_i.
 \end{aligned} \tag{4.1}$$

In the Range Mapping, $RM(M)$, M is divided into I non-overlapping source segments, $s_i \in S = \{s_1, s_2, \dots, s_I\}$, i.e., $\bigcup_{i=1}^I (s_i) = M$ and $\bigcap_{i=1}^I (s_i) = \emptyset$. Each and every source segment s_i is then

multiplied with an appropriate constant, G_i , so that they are mapped to the destination segments d_i that are confined within a range, termed as the Focus Range (F). Hence, the range mapping is defined as below.

$$\begin{aligned} RM(M): S \times G &\rightarrow D = \forall_i, s_i \times G_i \rightarrow d_i, \\ G_i &\in G = \{G_1, G_2, \dots, G_I\}, \\ d_i &\in D = \{d_1, d_2, \dots, d_I\}, \end{aligned} \quad (4.2)$$

and

$$F = \cup_{i=1}^I (d_i). \quad (4.3)$$

After mapping, the new range of approximation is the focus range F which is a range within the Mitchell range M as illustrated in the example with $I = 4$ in Fig. 4-1. Note that the destination segments in the focus range F may overlap and F is smaller than M . Hence, the Mitchell range, $M = [1,2)$ is now compressed to F .

Mapping of I segments requires I constants G_i . The selection of G_i must adhere to the following condition.

$$\forall_i: G_i^L \leq G_i \leq G_i^U \quad (4.4)$$

where G_i^U and G_i^L are the upper and lower bound ratios for the segment s_i as defined below.

$$\begin{aligned} G_i^U &= \frac{U_F}{U_i} \\ G_i^L &= \frac{L_F}{L_i} \end{aligned}$$

where U_F and L_F are the upper and lower bound values respectively of the focus range F while U_i and L_i are the upper and lower bound values of the segment s_i . U_F and L_F are indicated in Fig. 4-1, where U_i and L_i for s_2 (i.e. U_2 and L_2) are also labeled. To compress the Mitchell range, U_F and L_F must stay within the Mitchell range, i.e.,

Fig. 4-1: A range mapping example

```

find_F(S, I, Q)
{
  constant size_M = 2 - 1 = 1;    // M = [1,2)
  constant step = 0.01;          // step size
  for q = 1 to Q do                // searching from Q points
  {
    L_F = U_F = 1 + q × size_M / (Q + 1);    // one point
    for i = 1 to I do Gi =  $\phi$ ;
    while any Gr =  $\phi$  do
    {
      L_F = L_F - step;          // expand range by step
      U_F = U_F + step;          // size at either side
      Fq = [L_F, U_F];
      foreach si find all Gi satisfying condition (4.4) for Fq; }
    repeat
    {
      U_F = U_F - step;          // U_F could be too high
      foreach si find all Gi satisfying condition (4.4) for Fq; }
    until any Gi =  $\phi$ ;
    U_F = U_F + step;
    repeat
    {
      L_F = L_F + step;          // L_F could be too low
      foreach si find all Gi satisfying condition (4.4) for Fq; }
    until any Gi =  $\phi$ ;
    L_F = L_F - step; }          // next q
  return all Fq;                // return Q ranges
}

```

Fig. 4-2: Algorithm for finding focus ranges

$$1 \leq L_F < U_F \leq 2.$$

In addition to the condition (4.4), the selection of G_i , hence the focus range F , is governed by the following considerations for circuit simplicity and approximation accuracy.

- (i) The constants G_i should comprise only, and as few as possible, power-of-two terms (POTTs) so that only adders (and no multiplier) are required and the number of them is kept minimum.
- (ii) The focus range F should be as small as possible to enhance the effectiveness of the piecewise linear approximation.

In this work, the algorithm **find_F(S, I, Q)** in Fig. 4-2 is proposed to search for the focus range F . This algorithm adheres to the condition (4.4) and the above two considerations. **find_F(S, I, Q)** takes as input the set of I source segments, $S = \{s_1, s_2, \dots, s_I\}$, and starts the search from Q uniformly distributed points in the range M . For each point $q \in \{1, 2, \dots, Q\}$, a range is established by expanding at both sides of q with a small step size of 0.01 one at a time. The expansion is stopped when at least one constant G_i satisfying (4.4) is found for each and every source segment s_i . In this work, due to the consideration (i), the G_i satisfying (4.4) is limited to have two POTTs at most during the search. Any G_i values having more than two POTTs are rejected even if they satisfy (4.4). As a result, a focus range is found, adhering to the condition (4.4) and the above two considerations at the same time. As the range is expanded at both ends at the same time, the range

found at this stage could be overshoot at one end or the other. Hence, it is contracted from one end then the other until (4.4) fails. In this way, a compact range located at point q is found. Since the search starts from Q points, Q focus ranges will be found.

The algorithm in Fig. 4-2 is illustrated here with a simplified example. Without losing generality, let's assume that the Mitchell range M is divided equally into four segments, $s_{i=1}, s_{i=2}, s_{i=3}, s_{i=4}$, and the first q point is expanded at $m_2 = 1.5$ (i.e. Starting point at 0.5) with a step size of 0.01. At step 1, the first range (termed as F_q in Fig. 4-2) of 0.02 to be checked is centered at $m_2 = 1.5$ and $F_q = [L_F, U_F) = [1.49, 1.51)$. Considering the first segment $s_{i=1} = [L_i, U_i) = [1.00, 1.25)$ first, $G_i^U = \frac{U_F}{U_i} = \frac{1.51}{1.25} = 1.208$ and $G_i^L = \frac{L_F}{L_i} = \frac{1.49}{1.00} = 1.49$ are computed. As G_i^L is greater than G_i^U , it is impossible to find a constant G_i that satisfies (4.4) for $s_{i=1}$. The same $F_q = [1.49, 1.51)$ is used to find G_i^U and G_i^L for the other segments, $s_{i=2}, s_{i=3}$ and $s_{i=4}$ to find the corresponding G_i . The following two steps (S1 and S2) are repeated until at least one G_i satisfying (4.4) is found for every segment, s_i .

(S1) Range F_q is expanded with a step size of 0.01 at both sides, i.e., $[L_F, U_F) = [L_F - 0.01, U_F + 0.01)$.

(S2) $G_i^U = \frac{U_F}{U_i}$ and $G_i^L = \frac{L_F}{L_i}$ are computed for $i = \{1, 2, 3, 4\}$ and (4.4) is checked.

When the expansion stops, a focus range F_q (not fully optimized as the above steps are only part of the algorithm in Fig. 4-2) is found for the point q , i.e. the expansion of q results in a focus range. The constants G_i for the corresponding segments $s_{i=1}, s_{i=2}, s_{i=3}, s_{i=4}$ are obtained at the same time because G_i are determined at each expansion of F_q .

In this work, **find_F(S, I, Q)** was run with $Q = 4, 9, 19$ and 24 each with $I = \{2, 3, 4\}$. These Q values are chosen such that the start points are distributed evenly in the range M , i.e., M is divided into 5, 10, 20 and 25 equal portions respectively. $I = 1$ was not considered as the only possible destination segment would be the same as the source segment and the Mitchell range M would not be compressed. The search was not performed for I values larger than 4 as they will lead to higher hardware complexity. For $I = 2$ and 4, the source segments are derived from dividing the Mitchell range M equally. For $I = 3$, M is divided with only two MSBs of m used for segment selection and hence, 3 different sets of source segments S are obtained, i.e., $\{s_1, s_2, s_3\} = \{[1, 1.50), [1.50, 1.75), [1.75, 2)\}$, $\{[1, 1.25), [1.25, 1.75), [1.75, 2)\}$ and $\{[1, 1.25), [1.25, 1.50), [1.50, 2)\}$. From all the above runs, a total of exactly 20 distinct focus ranges were obtained, of which 6 were

TABLE 4-1: FOCUS RANGES FOUND FOR $I=4$

Start points			F_q	Size of F_q
$Q=4$	$Q=9$	$Q=19$		
–	0.10	0.05 0.10 0.15	[1.03,1.32)	0.29
0.20	0.20	0.20	[1.06,1.33)	0.27
–	–	0.25	[1.09,1.41)	0.32
–	0.30	0.30	[1.12,1.50)	0.38
0.40	0.40 0.50	0.35 0.40 0.45 0.50 0.55	[1.25,1.57)	0.32
0.60 0.80	0.60 0.70 0.80 0.90	0.60 0.65 0.70 0.75 0.80 0.85 0.90 0.95	[1.50,1.88)	0.38

obtained with $I = 4$ as listed in Table 4-1. Note that no additional focus range is found for $Q = 24$ and hence, the search was not performed for larger Q values. For each focus range found, a piecewise approximation with the best accuracy and hardware complexity is determined in the way described in Section 4.3. The focus range corresponding to the best piecewise approximation is found to be $F_q = [1.25, 1.57)$ listed in Table 4-1. Note that the size of focus range is only one of the factors that impact the effectiveness of the piecewise approximation function in approximating the logarithm curve. The best focus range $F_q = [1.25, 1.57)$ found in this case is not the smallest as can be seen from Table 4-1. The other factor is the location of the focus range that determines the variation of the gradient of the logarithm curve as explained later in this section. Also note that it is possible to have a portion of the focus range resided outside the Mitchell range, i.e., $F \not\subseteq M$, as discussed in Section 4.5.

When searching for the focus range, there are cases that more than one possible G_i values can be found. Table 4-2 shows one of such cases where G_2 has more than one valid values for the selected F_q . The mappings are further analyzed to determine the G_2 value. Fig. 4-3(a) and (b) show respectively the mappings with $G_2 = 2^0$ and $G_2 = 2^0 + 2^{-5}$ and the corresponding destination segments, where m_2 is the new Mitchell fraction corresponding to the focus range F . The mapping with $G_2 = 2^0$ in Fig. 4-3(a) was found to be a better choice for two reasons. First, $G_2 = 2^0$ contains only one POTT and hence leads to slightly lower hardware complexity. More

Fig. 4-3: Destination segments d_i obtained from range mapping with G_i from Table 4-2: (a) $G_2 = 2^0$; (b) $G_2 = 2^0 + 2^{-5}$.

TABLE 4-2: SELECTED AND COMPUTED FOCUS RANGE

Selected F_q	[1.25, 1.57]			
i	1	2	3	4
s_i	[1.00, 1.25)	[1.25, 1.50)	[1.50, 1.75)	[1.75, 2.00)
G_i^U	1.256	1.047	0.897	0.785
G_i^L	1.250	1.000	0.833	0.714
Possible G_i	$2^0 + 2^{-2}$	$2^0 + 2^{-5}$ $2^0 + 2^{-6}$ $2^0 + 2^{-7}$ $2^0 + 2^{-8}$ $2^0 + 2^{-9}$ $2^0 + 2^{-10}$ $2^0 + 2^{-11}$ $2^0 + 2^{-12}$ 2^0	$2^0 \cdot 2^{-3}$	$2^0 \cdot 2^{-2}$
Selected G_i	$2^0 + 2^{-2}$	2^0	$2^0 \cdot 2^{-3}$	$2^0 \cdot 2^{-2}$
Computed F	[1.25, 1.5625)			

importantly, the resultant mapping leads to fewer m_2 values for the 16 small N values (1, 2, 3, 4, ..., 16), which are termed here as the critical points. The mapping in Fig. 4-3(a) has 5 m_2 values (marked with 'x') while that of Fig. 4-3(b) has 6 for the 16 critical points. The mapping with fewer m_2 values for the critical points is preferred as explained later in this section. The other G_2 s are not selected for the same reasons. With the G_i values determined, the final focus range $F = [1.25, 1.5625)$ is computed from (4.2) and (4.3).

Effectively, the original range of $(1 + m)$, i.e., the Mitchell range = $[1, 2)$, is mapped and compressed to the focus range $F = [1.25, 1.5625)$. After range mapping, the following properties are associated with the range of m_2 :

- (i) m_2 has a smaller range than m ;
- (ii) $\log_2(1 + m_2)$ is more linear than $\log_2(1 + m)$;
- (iii) There are fewer Mitchell fractions.

Fig. 4-4: Focus range

Fig. 4-5: Gradient g of the $\log_2(1 + m_2)$ curve

Property (1) is obvious as the range is compressed to less than one third of the original range as shown in Fig. 4-4, where the dotted portion is outside the focus range and not required to be approximated as all $(1 + m)$ values will fall within the focus range after mapping. Property (2) can be observed from Fig. 4-5 which shows the variation of the gradient g of the $\log_2(1 + m_2)$ curve with the pair $(1 + m_2, g)$ denoting the value of $(1 + m_2)$ and g at each point. It is evident that the gradient variation is only $(1.15 - 0.92 = 0.23)$ within the compressed focus range $F = [1.25, 1.5625]$. The gradient variation would have been $(1.44 - 0.72 = 0.72)$ should the range mapping not be applied. Property (3) can be observed from Fig. 4-3(a), where the inputs $N = 1, 2, 4, 5, 8, 10$ and 16 share the same Mitchell fraction value ($m_2 = 0.25$) and $N = 3, 6, 7, 12$ and 14 share $m_2 = 0.3125$ after mapping. There are other inputs N sharing m_2 values. Hence, there are fewer Mitchell fractions.

All the three properties facilitate the reduction of error and error percentage when piecewise linear approximation is used to obtain the $\log_2(1 + m_2)$ values, and hence the $\log_2(N)$ values. With Properties (1) and (2), the piecewise straight lines can match closer with the actual logarithm curve. With Property (3), the error reduction can be focused on the critical points to reduce the maximum error percentages which are often associated with the critical points due to their relatively smaller values of $\log_2(N)$ in the denominator of (4.7). Thus, significant reduction in the *MAEP* and *MAE* can be achieved when applying the 4-region piecewise linear approximation on

the new range as described in Section 4.3. Furthermore, when the proposed UMA in Chapter 3 is applied on the focus range F , the three advantageous properties stated earlier facilitate the search for an optimized piecewise linear approximation function and, hence, higher accuracy (15 bits – 18 bits) logarithmic converters can be obtained within practical period of time as described in Section 4.4.

4.3 Piecewise Linear Approximation by Heuristic Approach

After range mapping, $\log_2(1 + m_2)$ is approximated by the conventional Mitchell-based piecewise linear approximation. A new approximation function in the form of (2.8) is formulated based on the following criteria for each of the 20 focus ranges found by **find_F(S, I, Q)** for $Q = \{4, 9, 19\}$ with $I = \{2, 3, 4\}$.

- (i) The maximum number of POTTs in the coefficient C_r in (2.8) for each region is limited to one to save adders.
- (ii) The maximum number of different C_r values across all regions is limited to two to reduce the complexity of the MUX in front the adder (see the approximation block in Fig. 4-7).
- (iii) The maximum number of regions is limited to four to restrain the logics that are needed to control the block implementing the approximation constants (D_r in Fig. 4-7).
- (iv) Unlike the convention methods where the wordlength truncation of m_2 is performed after the formulation of the approximation function, the truncation is considered during the formulation of the approximation function as the truncation error may be used to offset the approximation error.

With the above criteria applied during the formulation of the piecewise linear approximation, it is found that the focus range $F = [1.25, 1.5625)$ presented in Section 4.2 leads to the approximation function with the highest accuracy. The corresponding approximation function, $LA_{4,Q}(1 + m_2)$, has 4 regions as given in (4.5), where the 4 regions are selected by 4 bits of m_2 (2nd to 5th MSB). Note that the 4 regions here are neither the same as nor related to the 4 destination segments in the range mapping stage in Section 4.2 and they are determined after the focus range is derived. In general, the number of regions used for the piecewise linear approximation of the focus range may or may not be the same as the number of segments for range mapping.

TABLE 4-3: VALUES FOR CONSTANTS D_1, D_2, D_3 AND D_4

Constants	Values
D_1	$\log_2(1 + 0.250) - 0.250 \times 1.125$
D_2	$\log_2(1 + 0.34375) - 0.34375 \times 1.000 + 0.5 \times 0.001911$
D_3	$\log_2(1 + 0.375) - 0.375 \times 1.000 + 0.5 \times 0.000588$
D_4	$\log_2(1 + 0.53125) - 0.53125 \times 1.000 - 0.5 \times 0.002101$

$$\begin{aligned}
& \text{LA}_{4,Q}(1 + m_2) \\
&= \begin{cases} m_2 + 2^{-3}m_{2MSB7} + D_1 & m \in [0.2500, 0.34375) \\ m_2 + D_2 & m \in [0.34375, 0.3750) \\ m_2 + D_3 & m \in [0.3750, 0.53125) \\ m_2 + D_4 & m \in [0.53125, 0.5625) \end{cases} \quad (4.5)
\end{aligned}$$

where m_{2MSB7} is the 7 MSBs of m_2 . D_1, D_2, D_3 and D_4 are constants determined empirically by Matlab simulations to minimize the error and error percentage due to the approximation of $\log_2(1 + m_2)$. It is worth noting that there are only two C_r (see (2.8)) values, 2^{-3} and 0, in the approximation functions (4.5), resulting in a simple MUX in front of the adder in the approximation circuit. The process to determine the constants consists of the following two main steps with D_1 as an illustrative example.

(i) In this region, m_2 has an effective range of 0.25 to 0.34375 and the starting point is 0.25. First, $D_{init,1}$ is determined by solving the equation $m_2 + 2^{-3}m_{2MSB7} + D_{init,1} = \log_2(1 + m_2)$ for $m_2 = 0.25$, i.e., to obtain a value for $D_{init,1}$ such that the straight line $m_2 + 2^{-3}m_{2MSB7} + D_{init,1}$ and the curve $\log_2(1 + m_2)$ intercepts at the starting point.

(ii) Then D_1 is obtained by manually adding a fine-tuning value to $D_{init,1}$ such that the straight line obtained in (i) is shifted vertically and Matlab simulation is performed to assess the accuracy in terms of error and error percentage.

The other constants are similarly obtained with the 2-step process and listed in Table 4-3. Consequently, the piecewise linear approximation of $\log_2 N$ in the proposed method is given as

$$\begin{aligned}
\log_2 N &= k + \log_2(1 + m_2) - \log_2 G_r \\
&\cong k + \text{LA}_{4,Q}(1 + m_2) - \log_2 G_r
\end{aligned} \quad (4.6)$$

where $\text{LA}_{4,Q}(1 + m_2)$ is determined with (4.5) and $(-\log_2 G_r)$ is a constant resulted from the range mapping as shown in (4.1). The constants D_r , for $r = \{1, 2, 3, 4\}$, from $\text{LA}_{4,Q}(1 + m_2)$ are added to $(-\log_2 G_r)$ and then truncated to only 11 bits. The summation of the constants is pre-computed and incurs no extra adder cost as discussed in Section 4.3.2.

4.3.1 Accuracy

The proposed method is modeled and simulated in Matlab with 32-bit integer input N . The double-precision logarithm function in Matlab is used to emulate the accurate values of $\log_2(N)$. Fig. 4-6 depicts the simulation results of the error percentage for the proposed RM method together with those from the existing methods shown in Fig. 2-4. Table 4-4 provides the accuracy comparisons among the proposed method and the well-known Mitchell-based methods in terms of error (E) and error percentage (EP), where

$$E = \log_2(1 + m) - \text{LA}_{R.MA\text{-based methods}}(1 + m)$$

for the existing Mitchell-based methods while

$$E = \log_2(1 + m) - [\text{LA}_{4.Q}(1 + m_2) - \log_2 G_r]$$

for the proposed range mapping method, and

$$EP = E / \log_2(N). \quad (4.7)$$

Table 4-4 is exactly the same as Table 2-2 except that a column on the proposed RM method is added. As mentioned in Section 2.1.11, among the existing Mitchell-based methods, the methods by Kim et al. and De Caro et al. achieve, respectively, the best results in the MAE of 0.002 and $MAEP$ of 0.032% with their 8-region approximations. The proposed RM method outperforms the best results by 15.0% for MAE and 25.0% for $MAEP$. It is worth noting that the proposed method is able to achieve smaller error and error percentage simultaneously with fewer regions due to the three properties resulted from the range mapping.

TABLE 4-4: ACCURACY COMPARISONS AMONG THE PROPOSED METHOD AND THE EXISTING MITCHELL-BASED METHODS

Methods	Mitchell [Mit62] 1962	Combet [Com65] 1965	Hall [Hal70] 1970	San- Gregory [San99] 1999	Abed [Abe03] 2003			Kim [Kim06] 2006	Li [Li08] 2008	Juang [Jua09] 2009	Guti- errez [Gut11] 2011	De Caro [DeC11] 2011			This work
Regions	1	4	4	2	2	3	6	8	1	2	4	2	4	8	4
MPE ($\times 10^{-3}$)	86.1	8.0	4.4	29.3 [#]	44.9 [#]	29.3 [#]	13.2 [#]	2.0 ^{##1}	43.0	37.0 [#]	7.2	22.9 [#]	16.0 [#]	2.8 [#]	1.6
MNE ($\times 10^{-3}$)	0	-6.2	-8.2	-28.0 [#]	-18.3 [#]	-20.8 [#]	-13.0 [#]	-1.5 [#]	-43.0	-9.8 [#]	-7.9	-28.6 [#]	-8.9 [#]	-2.5 [#]	-1.7
ER ($\times 10^{-3}$)	86.1	14.2	12.6	57.3 [#]	63.2 [#]	50.1 [#]	26.2 [#]	3.5 [#]	86.0	46.8 [#]	15.1	51.5 [#]	24.9 [#]	5.3 [#]	3.3
MPEP (%)	5.361	0.185	0.126	0.431	0.930	0.431	0.153	0.050	2.645 [#]	1.079	0.181 [#]	0.582	0.189	0.032 ^{*2}	0.024
MNEP (%)	0	-0.267	-0.781	-1.540	-0.554	-0.268	-0.154	-0.044	-4.304 [#]	-0.096	-0.793 [#]	-0.498	-0.114	-0.031	-0.023
EPR (%)	5.361	0.452	0.907	1.971	1.484	0.699	0.307	0.094	6.949 [#]	1.175	0.974 [#]	1.080	0.303	0.063	0.047

#: results not reported in any previous works; *1: best existing MAE ($=\max(|MPE|, |MNE|)$); *2: best existing MAEP ($=\max(|MPEP|, |MNEP|)$)

Fig. 4-6: Error percentage (y-axis) of the existing Mitchell-based methods and the proposed method (x-axis is N)

4.3.2 Hardware Complexity

The architecture of the proposed method is depicted in Fig. 4-7, where the approximation block is similar to the conventional Mitchell-based methods but with different approximation function. In addition, there is a block implementing the range mapping. The architecture with $n = 32$ was coded in Verilog HDL for functional simulation and synthesized by the Synopsys Design Compiler with the STM90nm CMOS technology library. For fair comparison of hardware complexity, the architectures in Fig. 9(e) of [Abe03], Fig. 6 of [Kim06] and Fig. 6 of [DeC11] are synthesized using the same tool and technology library. These methods were selected for complexity comparison because [Kim06] and [DeC11] have the best *MAE* and *MAEP* respectively while [Abe03] has the most efficient hardware implementation (see Table 3-8) given its accuracy performance. The area, delay and power dissipation at 100 MHz obtained from the synthesis are tabulated in Table 4-5. Like all the existing Mitchell-based methods, no low power

Fig. 4-7: Architecture of the proposed RM method

TABLE 4-5: SYNTHESIS RESULTS IN STM90NM CMOS PROCESS

	Abed [Abe03]	Kim [Kim06]	De Caro [DeC11]	This work	Improvement over		
					Abed	Kim	De Caro
A (μm^2)	1643	3565	2767	2987	-81.8%	16.2%	-8.0%
D (ns)	1.21	2.19	1.70	2.03	-67.8%	7.3%	-19.4%
P (mW)	0.502	1.798	1.071	1.414	-181.7%	21.4%	-32.0%
ADP	1988	7807	4704	6064	-205.0%	22.3%	-28.9%

design techniques were employed in the current implementation.

As shown in Table 4-5, the proposed RM method is more efficient in hardware implementation than [Kim06] with improvements of 16.2% and 7.3%, respectively, in area cost (A) and computation delay (D). This leads to 22.3% reduction in area-delay-product (ADP). The improvements are attributed to the range mapping which relaxes the hardware complexity of the approximation block as discussed below. With the aid of the search algorithm **find_F(S, I, Q)**, a feasible focus range is selected such that high accuracy can be achieved with only four approximation functions consisting of a total of only two different C_r values of 2^{-3} and 2^0 (see (4.5)). As such, the approximation functions can be realized efficiently using only two adders (one 10 bits and the other 11 bits), one 7-bit 2-to-1 MUX and some simple control logics in the ‘C’ module of Fig. 4-7, where 2 bits of m (m_{MSB2}) determine the segments and 4 bits of m_2 ($m_{2MSB2 \rightarrow 5}$) determine the regions in (4.5) for the ‘C’ and ‘CONSTANTS’ modules. Besides, the search algorithm also keeps the complexity of the range mapping block low by ensuring that the mapping constants G_i consist of only two power-of-two terms.

Compared to Abed and Siferd’s 6-region approximation [Abe03], the proposed method has 81.8% larger area and 67.8% longer delay, and hence an increased ADP of 205.0%. However, the proposed method achieves much higher accuracy than [Abe03] does with improvements of 87.1% and 84.4%, respectively, in MAE and $MAEP$ as shown in Table 4-4. As stated in [Kim06], the conversion error from [Abe03] and the other existing Mitchell-based methods is too large for certain applications like 3-D graphic. The proposed method is able to meet the high accuracy requirement with reduced hardware overhead as compared to [Kim06]. The proposed method incurs 8.0% higher area cost and 19.4% longer computation delay when compared to the 8-region approximation by De Caro et al. [DeC11]. However, the proposed method achieves higher accuracy as compared to [DeC11] with an improvement of 39.3% and 25.0%, respectively, in MAE and $MAEP$. To evaluate the performance of the methods more thoroughly with the factors accuracy, area and delay all included, three figures of merit are proposed for comparisons as discussed in the next subsection.

4.3.3 Figures of Merit

The following three Figures of Merit (*FoMs*) are proposed for a fairer and more thorough comparison among the proposed method, the method by De Caro et al. (best *MAEP*), the method by Kim et al. (best *MAE*) and the method by Abed and Siferd (lowest hardware cost for high accuracy approximation).

$$FoM1 = ADP \times MAE \times MAEP$$

$$FoM2 = ADP \times MAEP$$

$$FoM3 = ADP \times MAE$$

All three *FoMs* include *ADP* but each considers different accuracy performance metric(s). This is to assess if the hardware and delay overhead introduced by the proposed method in improving the accuracy performance metric(s) is justifiable.

As shown in Table 4-6, the proposed range mapping method has better *FoM1*, *FoM2* and *FoM3* by 93.9%, 52.5% and 60.7% respectively when compared to Abed and Siferd's method, by 68.3%, 62.7% and 34.0% when compared to Kim et al.'s method, and by 41.3%, 3.3% and 21.7% when compared to the method by De Caro et al. It is evident that the *ADP* overhead introduced in achieving higher accuracy by the proposed method is justifiable.

TABLE 4-6: FIGURES OF MERIT

	Abed [Abe03]	Kim [Kim06]	De Caro [DeC11]	This work	Improvement over		
					Abed	Kim	De Caro
<i>FoM1</i>	4.041	0.781	0.421	0.247	93.9 %	68.3%	41.3%
<i>FoM2</i>	306.2	390.4	150.5	145.5	52.5%	62.7%	3.3%
<i>FoM3</i>	26.24	15.61	13.17	10.31	60.7%	34.0%	21.7%

4.4 Piecewise Linear Approximation by UMA

As described in the previous section, the approximation of $\log_2(N)$ using the proposed range mapping (RM) method is formulated as shown in (4.6) and re-written in (4.8).

$$\begin{aligned}
\log_2 N &= k + \log_2(1 + m) \\
&= k + \log_2(1 + m_2) - \log_2 G_i \\
&\cong k + m_2 + C_r \cdot m_{2MSB}^* + (D_r - \log_2 G_i) \\
&= k + LA_{R.I. \#}(1 + m_2).
\end{aligned} \tag{4.8}$$

In other words,

$$\begin{aligned}
\log_2(1 + m) &\cong \text{LA}_{R,I,\#}(1 + m_2) \\
&= m_2 + C_r \cdot m_{2MSB}^* + (D_r - \log_2 G_i) \\
&= m_2 + C_r \cdot m_{2MSB}^* + D_{r,i}
\end{aligned} \tag{4.9}$$

where R is the number of regions of approximation, $I = 4$ is the number of segments that the Mitchell range M is divided into for mapping, G_i are the mapping constants given in Table 4-2, and $D_{r,i} = D_r - \log_2 G_i$ is a new constant. It can be seen that (4.9) is the same as (3.1) except that $m_2 \in F = [1.25, 1.5625)$ in (4.9) is mapped from $m \in M = [1, 2)$ of (3.1), and that there are four constants $D_{r,i} (i \in \{1, 2, 3, I\})$ for each region, g_r , as reflected in Fig. 4-7. As such, with necessary changes made as described in Section 4.4.1 below, the proposed UMA in Chapter 3 is applied on the focus range $F = [1.25, 1.5625)$ to obtain an optimized piecewise linear approximation function, $\text{LA}_{R,I,\#}(1 + m_2)$, for approximating $\log_2(1 + m)$, and hence $\log_2(N)$, for higher accuracy requirements (>14 bits).

4.4.1 Modifications to UMA for Focus Range

The following modifications are performed to the UMA method so as to apply it on the focus range $F = [1.25, 1.5625)$ obtained in the range mapping method.

i) The function **divideFailRegion**(g_r), which divides f-regions into two new-regions until all regions become s-regions described in Section 3.5, needs to consider the change of the range approximation to focus range F as shown in the example in Fig. 4-8. It is important to highlight that, unlike the cases in Section 3.5 where the two new regions divided from a f-region are equal size, the two new regions divided from f-regions within the focus range may not be equal in size as illustrated in the example in Fig. 4-8. In this example, the two new regions g_1 and g_2 in step (2) divided from the f-region ($g_1 = F = [1.25, 1.5625)$) of step (1) are having different sizes of 0.25 and 0.0625, respectively.

ii) The error analysis described in Section 3.7 is performed for each segment i of each region $g_r = [L_r, U_r)$ within the focus range $F = [1.25, 1.5625)$ with the modifications as described below.

Fig. 4-8: Divisions of regions

Here the piecewise straight line (4.9) is used instead of (3.13) for the accuracy check. Prior to the accuracy check, all the values of N which possess the m values that are mapped to the m_2 values that fall within the region g_r are determined. Correspondingly, the mapping constant G_i and segment i values of each N are known. Then, the accuracy check is performed by computing (4.11) and (4.12) to determine, respectively, the values $D_{r.i.maxl}$ and $D_{r.i.minu}$ for the constant $D_{r,i}$ in (4.9) for a segment i within the region g_r . If the maximum value of the $D_{r,i}$ lower bound ($D_{r.i.maxl}$) is less than or equal to the minimum value of the $D_{r,i}$ upper bound ($D_{r.i.minu}$), i.e., $D_{r.i.maxl} \leq D_{r.i.minu}$, there exists a constant $D_{r,i}$ such that (4.10) holds for all N , i.e. each and every EP (error percentage) is less than Acc and the accuracy requirement is satisfied. Hence, the corresponding gradient value C_r is accepted. Otherwise, the gradient value is rejected.

$$|EP| = \left| \frac{(\log_2(1+m) - LA_{R.I.\#}(1+m_2))}{\log_2(N)} \cdot 100 \right| \leq Acc \quad (4.10)$$

$$\Rightarrow -\frac{Acc}{100} \leq \frac{(\log_2(1+m) - (m_2 + C_r \cdot m_{2MSB}^* + D_{r,i}))}{\log_2(N)} \leq \frac{Acc}{100}$$

$$D_{r.i.l} = \log_2(1+m) - m_2 - C_r \cdot m_{2MSB}^* - \frac{Acc}{100} \cdot \log_2(N)$$

$$D_{r.i.maxl} = \max_{\forall N} D_{r.i.l} \quad (4.11)$$

$$D_{r.i.u} = \log_2(1+m) - m_2 - C_r \cdot m_{2MSB}^* + \frac{Acc}{100} \cdot \log_2(N)$$

$$D_{r.i.minu} = \min_{\forall N} D_{r.i.u} \quad (4.12)$$

iii) As shown in Fig. 4-7, two MSBs of m , m_{MSB2} , are used to determine the segment i , in addition to $m_{2MSB2 \rightarrow 5}$ that are used to determine the region r , to select the constant $CONSTANTS = D_r - \log_2 G_i$. Taking this into consideration, the architectural model of the approximation circuit that implements (4.9) is obtained from modifying the model in Fig. 3-1 and given in Fig. 4-9. The main modification is that the Control Logic block of Fig. 3-1 is separated into two blocks, Control Logic (1) and Control Logic (2), in Fig. 4-9. Control Logic (1) is the same as Control Logic of Fig. 3-1 except that it does not determine SEL anymore. On the other hand, Control Logic (2) is a new control logic block which takes in m_{MSB2} in addition to m_{2MSBB} to determine the select signal, SEL , for selecting the constant $D = D_r - \log_2 G_i$.

4.4.2 Performances Evaluation

Applying the changes described in Section 4.4.1, the proposed UMA in Chapter 3 is applied on the focus range $F = [1.25, 1.5625)$ to obtain designs with the accuracy increased bit by bit starting from 14 bits as the proposed UMA has obtained designs with accuracy up to 14 bits as presented in Chapter 3. Table 4-7 lists the time needed to obtain the designs, the estimated and the

Fig. 4-9: Generic architectural model for approximation circuit

TABLE 4-7: ESTIMATED AND SYNTHESIS RESULTS OF THE HARDWARE MERITS

Accuracy	Time (s)	Estimation			Synthesis		
		A	τ	ADP	A (μm^2)	τ (ns)	ADP
14 bits	886	2208	111	245088	17660	3.26	57572
15 bits	1514	2744	113	310072	19695	3.26	64206
16 bits	58475	3122	121	377762	20630	3.56	73443
17 bits	5637	4196	121	507716	23410	3.52	82403
18 bits	652152	4954	131	648974	23147	3.84	88884

synthesis results. The computation time increases significantly for accuracy higher than 17 bits. For accuracy of 19 bits, the search took too long and it was abandoned.

The designs obtained for accuracies from 14 bits to 18 bits are given in Table 4-8. They are modeled and simulated in Matlab for 32-bit input N to verify the accuracy. As mentioned in Section 3.3.1, if the accuracy is not met in the 32-bit simulations, UMA is re-executed for the same accuracy with larger wordlength of N in the function **analyzeErr**. This situation occurs for accuracy requirements of 16, 17 and 18 bits where UMA is re-executed with $n_e = 14, 11$ and 13 ($N_{maxerr} = 10303, 1253$ and 5135), respectively, in the error analysis (see Section 3.3.1). Beside Matlab modeling and simulations, they are modeled in Verilog HDL for functional verification and synthesized in STM 90 nm CMOS technology by Synopsys Design Compiler. The estimated and synthesis results for the area (A), delay (τ) and area-delay product (ADP) are included in Table 4-7 and plotted in Fig. 4-10 (a) and (b) respectively. Similar to Fig. 3-8 (c), both the results are normalized to their maximum values and plotted in Fig. 4-10(c). It is evident that the two

curves in Fig. 4-10(c) have well-matched trends, demonstrating the efficacy of the estimation method.

Fig. 4-10: ADP of the approximation functions

TABLE 4-8: DESIGNS OBTAINED BY APPLYING THE UMA ON FOCUS RANGE F FOR ACCURACIES OF 14 BITS TO 18 BITS

Acc (bits)	Designs	T	g_r	e_r	$(f_r - \log_2(G_i)) (\times 2^{-T})$			
					$i = 1$	$i = 2$	$i = 3$	$i = 4$
14	$LA_{7,R}(1+m)$ (MAEP = 0.00588%)	13	[1.250000,1.265625)	2^{-3}	-2304	334	0	0
			[1.265625,1.281250)	2^{-3}	-2302	334	0	0
			[1.281250,1.312500)	2^{-3}	-2302	336	0	0
			[1.312500,1.375000)	$2^{-4} + 2^{-6}$	-2182	454	2032	3854
			[1.375000,1.437500)	2^{-5}	-2040	596	2174	3996
			[1.437500,1.500000)	$2^{-7} - 2^{-5}$	-2100	536	2116	3936
			[1.500000,1.562500)	-2^{-4}	-2194	1	2020	1
15	$LA_{11,S}(1+m)$ (MAEP = 0.00302%)	13	[1.2500000,1.2578125)	2^{-3}	-2304	333	0	0
			[1.2578125,1.2656250)	2^{-3}	-2301	335	0	0
			[1.2656250,1.2812500)	2^{-3}	-2301	335	0	0
			[1.2812500,1.3125000)	$2^{-3} - 2^{-6}$	-2391	247	0	0
			[1.3125000,1.3437500)	$2^{-4} + 2^{-5}$	-2223	413	1992	3814
			[1.3437500,1.3740000)	2^{-4}	-2137	499	2079	3901
			[1.3740000,1.4062500)	2^{-5}	-2041	596	2175	3997
			[1.4062500,1.4375000)	2^{-6}	-1988	649	2228	4049
			[1.4375000,1.4687500)	0	-1932	704	2282	4104
			[1.4687500,1.5000000)	-2^{-5}	-2068	568	2146	3968
16	$LA_{13,T}(1+m)$ (MAEP = 0.00150%)	14	[1.25000000,1.25390625)	2^{-3}	-4608	667	0	0
			[1.25390625,1.25781250)	2^{-3}	-4604	668	0	0
			[1.25781250,1.26562500)	2^{-3}	-4604	670	0	0
			[1.26562500,1.28125000)	2^{-3}	-4602	672	0	0
			[1.28125000,1.31250000)	$2^{-3} + 2^{-7} - 2^{-6}$	-4819	455	0	0
			[1.31250000,1.34375000)	$2^{-3} - 2^{-5} - 2^{-7}$	-5044	230	3386	0
			[1.34375000,1.37500000)	2^{-4}	-4274	1000	4156	7800
			[1.37500000,1.40625000)	$2^{-5} + 2^{-7}$	-4130	1144	4300	7944
			[1.40625000,1.43750000)	2^{-6}	-3976	1299	4456	8099
			[1.43750000,1.46875000)	-2^{-7}	-3934	1340	4496	8140
			[1.46875000,1.50000000)	-2^{-5}	-4138	1136	4293	7936
			[1.50000000,1.53125000)	$-2^{-5} - 2^{-6}$	-4265	1	4166	1
			[1.53125000,1.56250000)	-2^{-4}	-4387	1	4166	1
17	$LA_{21,U}(1+m)$ (MAEP = 0.000760%)	15	[1.250000000,1.251953125)	2^{-3}	-9216	1333	0	0
			[1.251953125,1.253906250)	2^{-3}	-9212	1337	0	0
			[1.253906250,1.257812500)	2^{-3}	-9210	1337	0	0
			[1.257812500,1.265625000)	$2^{-3} + 2^{-6}$	-9342	1208	0	0
			[1.265625000,1.281250000)	$2^{-3} + 2^{-7}$	-9272	1276	0	0
			[1.281250000,1.296875000)	2^{-3}	-9202	1348	0	0
			[1.296875000,1.312500000)	$2^{-3} - 2^{-6}$	-9564	986	0	0
			[1.312500000,1.328125000)	$2^{-3} - 2^{-5}$	-9916	632	6945	14232
			[1.328125000,1.343750000)	$2^{-4} + 2^{-6}$	-8725	1824	8136	15424
			[1.343750000,1.359375000)	2^{-4}	-8548	2001	8314	15602
			[1.359375000,1.375000000)	$2^{-4} - 2^{-8}$	-8628	1920	8233	15521
			[1.375000000,1.390625000)	$2^{-4} - 2^{-6}$	-8870	1680	7992	15278
			[1.390625000,1.406250000)	2^{-5}	-8160	2388	8702	15988
			[1.406250000,1.421875000)	$2^{-6} + 2^{-7}$	-8056	2492	8806	16093
			[1.421875000,1.437500000)	2^{-7}	-7840	2708	9020	16308
			[1.437500000,1.453125000)	0	-7728	2820	9132	16420
			[1.453125000,1.468750000)	-2^{-7}	-7870	2679	8992	16279
			[1.468750000,1.484375000)	$2^{-7} - 2^{-5}$	-8398	2150	8462	15750
			[1.484375000,1.500000000)	$-2^{-5} - 2^{-7}$	-8405	2144	8457	15744
			[1.500000000,1.515625000)	$2^{-6} - 2^{-4}$	-9043	1	7819	1
			[1.515625000,1.531250000)	$-2^{-4} - 2^{-8}$	-8832	1	7819	1

TABLE 4-8 (CONTINUE)

Acc (bits)	Designs	T	g_r	e_r	$f_r (\times 2^{-T})$			
					$i = 1$	$i = 2$	$i = 3$	$i = 4$
18	$LA_{24,V}(1+m)$ ($MAEP = 0.000379\%$)	16	[1.2500000000,1.2509765625)	2^{-3}	-18432	2666	19536	30511
			[1.2509765625,1.2519531250)	2^{-3}	-18430	2667	19536	30511
			[1.2519531250,1.2539062500)	2^{-3}	-18426	2670	19536	30511
			[1.2539062500,1.2578125000)	$2^{-3} + 2^{-6}$	-18684	2414	19536	30511
			[1.2578125000,1.2656250000)	$2^{-3} + 2^{-6}$	-18682	2416	19536	30511
			[1.2656250000,1.2812500000)	$2^{-3} + 2^{-7}$	-18544	2553	19536	30511
			[1.2812500000,1.2968750000)	$2^{-3} + 2^{-8} - 2^{-7}$	-18840	2258	19536	30511
			[1.2968750000,1.3125000000)	$2^{-3} - 2^{-6} - 2^{-8}$	-19305	1794	19536	30511
			[1.3125000000,1.3281250000)	$2^{-4} + 2^{-5}$	-17786	3312	19536	30511
			[1.3281250000,1.3437500000)	$2^{-4} + 2^{-6} + 2^{-8}$	-17536	3562	16188	30762
			[1.3437500000,1.3593750000)	$2^{-4} + 2^{-8}$	-17184	3912	16538	31114
			[1.3593750000,1.3750000000)	$2^{-4} - 2^{-7}$	-17418	3678	16304	30878
			[1.3750000000,1.3906250000)	$2^{-5} - 2^{-8}$	-16874	4223	16848	31424
			[1.3906250000,1.4062500000)	2^{-5}	-16318	4778	17404	31978
			[1.4062500000,1.4218750000)	$2^{-5} - 2^{-7} - 2^{-8}$	-16774	4325	16950	31524
			[1.4218750000,1.4375000000)	2^{-7}	-15791	5306	17932	32507
			[1.4375000000,1.4531250000)	0	-15458	5640	18625	32840
			[1.4531250000,1.4687500000)	$-2^{-7} - 2^{-8}$	-15878	5220	17845	32420
			[1.4687500000,1.4843750000)	$2^{-7} - 2^{-5}$	-16798	4299	16926	31500
			[1.4843750000,1.5000000000)	$-2^{-5} - 2^{-8}$	-16681	4417	17042	31616
			[1.5000000000,1.5156250000)	$2^{-6} - 2^{-4}$	-18086	1	15637	1
			[1.5156250000,1.5312500000)	$2^{-7} - 2^{-4}$	-17820	1	15904	1
			[1.5312500000,1.5468750000)	-2^{-4}	-17546	1	1	1
			[1.5468750000,1.5625000000)	$-2^{-4} - 2^{-7}$	-17779	1	1	1

4.5 Discussions on Additional Mapping Option

4.5.1 Equivalence Property

As mentioned in Section 2.1.1, all the MI s have the same Mitchell range M , and are differentiated by only the characteristic k value. For example, the points at 2.65625, 5.3125 and 10.625 in $MI_1 = [2, 4)$, $MI_2 = [4, 8)$ and $MI_3 = [8, 16)$ have k values of 1, 2 and 3 respectively, but they have the same $(1+m)$ value of $1.328125_{10} = 1.010101_2$. In other words, despite their different MI sizes ($4 - 2 = 2$, $8 - 4 = 4$ and $16 - 8 = 8$ respectively), they have the same range of approximation. In general, for a value N_i in a Mitchell interval $MI_i = [2^i, 2^{i+1})$, there always exists a corresponding value $N_j = 2^{(i-j)}N_i$ in another Mitchell interval $MI_j = [2^j, 2^{j+1})$ such that $i \neq j$ and N_i and N_j have the same $(1+m)$ value. This characteristic is termed here as the Equivalence Property. Should the $(1+m)$ value be greater than 2 (or less than 1), it can be substituted with a value within the range of $[1, 2)$ by dividing (multiplying) it with 2^c and then adding (subtracting) c to (from) k . For example, $k + \log_2(1+m) = 2 + \log_2(2.25)$ is equal to

$3 + \log_2(1.125)$ after 2.25 is replaced by 1.125 with $c = 1$ added to $k = 2$. Equivalence property allows the mapping of a source segment to a destination segment outside its MI as discussed next.

4.5.2 Additional Mapping Option

After range mapping, the range of approximation is the focus range $F \subset M = [1, 2)$ as in Fig. 4-1. The location of F is determined by the constants G_i that place the destination segments d_i within the Mitchell range M . It is possible to choose the G_i values such that one or more destination segments extend beyond or completely reside outside the Mitchell range. Fig. 4-11 illustrates such a case where the destination segment d_4 has a part, C , resides outside the Mitchell range, i.e., $U_F > 2$ and $F = B \cup C \not\subseteq M$. By applying equivalence property, C can be substituted by its equivalence, $A = C/2$ in $[1, 2)$. Effectively, the new range of approximation is now $F_{eff} = A \cup B$, termed as the effective focus range.

The mathematical formulation is similar to (4.1), i.e.,

$$\begin{aligned} \log_2 N &= k + \log_2(1 + m) \\ &= k + \log_2[G_i(1 + m)] - \log_2 G_i \\ &= k + \log_2(1 + m_2) + c - \log_2 G_i \end{aligned} \quad (4.13)$$

where $c = 1$ when $1 + m_2$ is within A and $c = 0$ otherwise.

Formulation (4.13) leads to higher hardware cost than (4.1) does due to more complex control logic. For the example in Fig. 4-11, $1 + m_2 \in B$ is the direct output of the adder implementing the mapping whereas $1 + m_2 \in A$ is the adder output right shifted by one bit (divided by 2). Therefore, this mapping option is not considered in the current work but could be exploited in future work for searching alternative focus ranges to further improve conversion accuracy.

Fig. 4-11: Range mapping based on Equivalence Property

4.6 Summary

In this chapter, a novel approach named range mapping for high accuracy Mitchell-based logarithmic converters is presented. The proposed method compresses the range of approximation to less than one third of the Mitchell range. A search algorithm is developed to determine the compressed range such that high accuracy approximation can be achieved with minimum hardware and delay overhead. After range mapping, the compressed range is smaller, more linear and has fewer Mitchell fractions. With these properties, when a newly proposed 4-region piecewise linear approximation is developed by heuristic approach for the compressed range, the proposed method is able to reduce the maximum absolute error and maximum absolute error percentage of the binary logarithm computation by 15.0% and 25.0% respectively when compared to the best existing results obtained by Kim et al. [Kim06] and De Caro et al. [DeC11] respectively. The proposed figures of merit indicate that the *ADP* overhead introduced by the proposed method in achieving high accuracy is justifiable. Therefore, when the proposed UMA is applied on the compressed range, the three advantageous properties facilitate the search for an optimized piecewise linear approximation function and, hence, higher accuracy (15 – 18 bits) logarithmic converters are obtained within feasible period of computation time.

CHAPTER 5. A Memory Efficient Tables-and-Additions Method for Accurate Computation of Elementary Functions

5.1 Overview

The table-based tables-and-additions methods are studied for high accuracy computation of elementary functions which include binary logarithm and it is observed that the table (memory) size in the Add-Table Lookup-Add (ATA) method [Won95] is not optimized as a result of applying the central difference formulation (CDF) to approximate the first derivatives in the Taylor series of the function to be evaluated. Based on this observation, a new method named the integrated Add-Table Lookup-Add (iATA) method is proposed to abandon the CDF, leading to a new approximation formulation and high memory efficiency. Also, three additional techniques, specifically the carry select technique (CST), the symmetry property exploitation (SPE) and the unequal partitioning of input with the aid of error analysis, are integrated in the formulation to achieve further memory reduction. As such, the proposed iATA, features four techniques, (A) Approximation without CDF, (B) CST, (C) SPE and (D) Unequal partitioning of input. Their features and concepts are discussed below while the detailed mathematical formulations are presented in Section 5.2.

5.1.1 Approximation without CDF

ATA [Won95] evaluates the elementary functions by applying the Taylor series approximation (2.25), which contains the first derivative

$$t_1 = f^{(1)}(X_0 + \lambda X_1)(\lambda^2 X_2).$$

ATA applies the CDF with the following two terms

$$\begin{aligned} t_{1.1} &= \frac{\lambda}{2} f(X_0 + \lambda X_1 + \lambda X_2) \\ t_{1.2} &= \frac{\lambda}{2} f(X_0 + \lambda X_1 - \lambda X_2) \end{aligned} \tag{5.1}$$

in (2.28) to obtain t_1 . The two tables, a_1 and a_2 , which implement $t_{1.1}$ and $t_{1.2}$, are 204800 bits each and they constitute the largest percentage (23.6% each) of the total table size [Won95]. It is observed that both $t_{1.1}$ and $t_{1.2}$ contain the desired term t_1 in their Taylor series expansions (2.29). Hence a novel technique is proposed here to reduce the table size without compromising the accuracy by abandoning CDF and using only one of the two terms. In this case, $t_{1.2}$ and not $t_{1.1}$ is

used because the truncation of the higher order derivative terms in $t_{1,2}$ leads to smaller error as discussed later in Section 5.2.2. As a result, one of the two largest tables is eliminated. Similarly, the other first derivative

$$t_2 = f^{(1)}(X_0 + \lambda X_1)(\lambda^3 X_3)$$

in (2.25) is obtained from $t_{2,2}$ alone instead of both $t_{2,1}$ and $t_{2,2}$ given below.

$$\begin{aligned} t_{2,1} &= \frac{\lambda^2}{2} f(X_0 + \lambda X_1 + \lambda X_3) \\ t_{2,2} &= \frac{\lambda^2}{2} f(X_0 + \lambda X_1 - \lambda X_3) \end{aligned} \quad (5.2)$$

Hence, the 155648-bit table that implements $t_{2,1}$ is likewise eliminated.

5.1.2 Carry Prevention/Select Technique

The subtraction $X_0 + \lambda X_1 - \lambda X_2$ in $t_{1,2}$ (5.1) could generate a carry at the MSB position. In ATA, the carry bit is included as part of the input to the table implementing $t_{1,2}$, which leads to a larger table. The carry problem is investigated and two possible solutions are found for table size reduction. One solution is to prevent the carry generation, resulting in 1 bit less in the input to the table. The other is to split the table into two smaller ones and uses the carry to select one of the tables to use. Either solution leads to smaller total table size as detailed in Section 5.2.2. The same is applied to the subtraction $X_0 + \lambda X_1 - \lambda X_3$ in $t_{2,2}$ (5.2).

5.1.3 Symmetry Property Exploitation (SPE)

The concept for applying the SPE in iATA is the same as that of SBTM, in which the symmetrical property exists in the term $(\lambda_0 \lambda_1 X_2 - \delta_2)$ of its first derivative (2.35). In iATA, however, the first derivatives are obtained without the CDF and do not possess the symmetrical property. Although the second and higher derivatives in its formulation do possess the property implicitly, exploiting the property is not straightforward. A solution is proposed to uncover the property and the mathematical formulation is developed to integrate the SPE.

5.1.4 Unequal partitioning of input

ATA divides the input X into four equal parts (Fig. 2-5(a)), which inhibits the full utilization of the available error budget for table size optimization. With the aid of error analysis, iATA partitions the input into four parts of unequal wordlengths (Fig. 2-5(d)), facilitating the optimization of the wordlength of the parts that are used as the input of the large tables, and thus reducing the total table size.

The detailed mathematical formulations of the four techniques integrated in iATA are described and derived in the next section.

5.2 Mathematical Formulation

In the following subsections, the mathematical formulation for iATA is developed in stages in the sequence that its four techniques are introduced in Section 5.1. The formulation starts with the approximation without CDF. It is then extended to integrate the CST, followed by the integration of the SPE. The formulation is completed after the unequal partitioning of the input with the aid of error analysis is integrated. The complexity of the formulation grows after each technique is added in. For clearer mathematical derivations, symbols with the superscripts A, B, and C are used to denote mathematical terms related to the first three techniques and the superscripts have no mathematical meaning. E.g., D_0^A and D_3^C are two symbols used respectively in the derivations of the approximation without CDF and the SPE.

5.2.1 Approximation without CDF

In iATA, the n -bit input X is divided into four unequal parts as shown in Fig. 2-5(d). Mathematically,

$$X = X_0 + \lambda_0 X_1 + \lambda_0 \lambda_1 X_2 + \lambda_0 \lambda_1 \lambda_2 X_3$$

where $X_i = [0, 1 - 2^{-n_i}]$, $\lambda_i = 2^{-n_i}$ and $n = \sum n_i$ for $i = \{0, 1, 2, 3\}$. The Taylor series of the function $f(X)$ can be written as:

$$\begin{aligned} f(X) &= f(X_0 + \lambda_0 X_1 + \lambda_0 \lambda_1 X_2 + \lambda_0 \lambda_1 \lambda_2 X_3) \\ &= f(X_0 + \lambda_0 X_1) + \\ &\quad f^{(1)}(X_0 + \lambda_0 X_1)(\lambda_0 \lambda_1)(X_2) + f^{(1)}(X_0 + \lambda_0 X_1)(\lambda_0 \lambda_1 \lambda_2)(X_3) + \\ &\quad \frac{1}{2} f^{(2)}(X_0 + \lambda_0 X_1)[(\lambda_0 \lambda_1)^2 X_2^2 + 2(\lambda_0 \lambda_1)^2 (\lambda_2) X_2 X_3 + \\ &\quad (\lambda_0 \lambda_1 \lambda_2)^2 X_3^2] + \dots \end{aligned} \quad (5.3)$$

$f(X)$ is first approximated by

$$\tilde{f}(X)_{iATA} = D_0^A + D_1^A + D_2^A \quad (5.4)$$

where D_0^A , D_1^A and D_2^A are given in (5.5), (5.6) and (5.8) respectively and described below. iATA abandons the CDF and uses only one term each in D_1^A (5.6) and D_2^A (5.8) to approximate the two first derivatives of $f(X)$, $f^{(1)}(X_0 + \lambda_0 X_1)(\lambda_0 \lambda_1 X_2)$ and $f^{(1)}(X_0 + \lambda_0 X_1)(\lambda_0 \lambda_1 \lambda_2 X_3)$. After

applying Taylor series expansions to D_1^A and D_2^A , it is found that (5.7) and (5.9) contain respectively two undesired terms $(-\lambda_a)f(X_0 + \lambda_0 X_1)$ and $(-\lambda_c \lambda_d)f(X_0 + \lambda_0 X_1)$. These two terms are cancelled out by the two extra terms $(\lambda_a + \lambda_c \lambda_d)f(X_0 + \lambda_0 X_1)$ in D_0^A (5.5). The added terms do not incur any extra hardware cost because they can be pre-computed and merged with $f(X_0 + \lambda_0 X_1)$. On the other hand, the higher order derivatives of D_1^A and D_2^A are error terms to be corrected.

$$D_0^A = (1 + \lambda_a + \lambda_c \lambda_d)f(X_0 + \lambda_0 X_1) \quad (5.5)$$

$$D_1^A = -\lambda_a f(X_0 + \lambda_0 X_1 - \lambda_b X_2) \quad (5.6)$$

$$\begin{aligned} &= -\lambda_a \{f(X_0 + \lambda_0 X_1) \\ &\quad -f^{(1)}(X_0 + \lambda_0 X_1)[(\lambda_b)X_2] \\ &\quad + \frac{1}{2}f^{(2)}(X_0 + \lambda_0 X_1)[(\lambda_b)^2 X_2^2] \\ &\quad - \frac{1}{6}f^{(3)}(X_0 + \lambda_0 X_1)[(\lambda_b)^3 X_2^3] \\ &\quad + \frac{1}{24}f^{(4)}(X_0 + \lambda_0 X_1)[(\lambda_b)^4 X_2^4] - \dots\} \end{aligned} \quad (5.7)$$

$$D_2^A = -\lambda_c \lambda_d f(X_0 + \lambda_0 X_1 - \lambda_e X_3) \quad (5.8)$$

$$\begin{aligned} &= -\lambda_c \lambda_d \{f(X_0 + \lambda_0 X_1) \\ &\quad -f^{(1)}(X_0 + \lambda_0 X_1)[(\lambda_e)X_3] \\ &\quad + \frac{1}{2}f^{(2)}(X_0 + \lambda_0 X_1)[(\lambda_e)^2 X_3^2] \\ &\quad - \frac{1}{6}f^{(3)}(X_0 + \lambda_0 X_1)[(\lambda_e)^3 X_3^3] + \dots\} \end{aligned} \quad (5.9)$$

where $\lambda_i = 2^{-n_i}$ for $i = \{a, b, c, d, e\}$. Note that n_i for $i = \{a, b, c, d, e\}$ are not wordlengths. They are purely constant parameters that emerge from the mathematical derivation in the approximation of the first derivatives by D_1^A and D_2^A . The determination of the values of n_i , and hence λ_i , is discussed in Section 5.2.4.

The initial approximation function $P_A(X)$ (5.12) for iATA is then obtained by applying the following three-step derivations.

(i) Assuming $\lambda_0 \lambda_1 = \lambda_a \lambda_b$ and $\lambda_0 \lambda_1 \lambda_2 = \lambda_c \lambda_d \lambda_e$, (5.4) is subtracted from (5.3) to obtain the difference between the function $f(X)$ and its approximation $\tilde{f}(X)_{iATA}$ as follows (Elaboration on the two assumptions is given in Section 5.3).

$$\begin{aligned}
& f(X) - \tilde{f}(X)_{iATA} \\
&= \frac{1}{2}f^{(2)}(X_0 + \lambda_0 X_1)[(\lambda_0 \lambda_1)^2 X_2^2 + 2(\lambda_0 \lambda_1)^2 (\lambda_2) X_2 X_3 + \\
&\quad (\lambda_0 \lambda_1 \lambda_2)^2 X_3^2] \\
&\quad + \frac{1}{2}f^{(2)}(X_0 + \lambda_0 X_1)[(\lambda_a)(\lambda_b)^2 X_2^2] \\
&\quad - \frac{1}{6}f^{(3)}(X_0 + \lambda_0 X_1)[(\lambda_a)(\lambda_b)^3 X_2^3] \\
&\quad + \frac{1}{24}f^{(4)}(X_0 + \lambda_0 X_1)[(\lambda_a)(\lambda_b)^4 X_2^4] \\
&\quad + \frac{1}{2}f^{(2)}(X_0 + \lambda_0 X_1)[(\lambda_c \lambda_d)(\lambda_e)^2 X_3^2] \\
&\quad - \frac{1}{6}f^{(3)}(X_0 + \lambda_0 X_1)[(\lambda_c \lambda_d)(\lambda_e)^3 X_3^3] + \dots
\end{aligned} \tag{5.10}$$

Ignoring the terms that have equal or higher order than λ_i^5 in (5.10), e.g., $\frac{1}{24}f^{(4)}(X_0 + \lambda_0 X_1)[(\lambda_a)(\lambda_b)^4 X_2^4]$ which has $(\lambda_a)(\lambda_b)^4$, the difference becomes:

$$\begin{aligned}
& f(X) - \tilde{f}(X)_{iATA} \\
&\approx \frac{1}{2}f^{(2)}(X_0 + \lambda_0 X_1)[(\lambda_0 \lambda_1)^2 X_2^2] \\
&\quad + \frac{1}{2}f^{(2)}(X_0 + \lambda_0 X_1)[(\lambda_a)(\lambda_b)^2 X_2^2] \\
&\quad - \frac{1}{6}f^{(3)}(X_0 + \lambda_0 X_1)[(\lambda_a)(\lambda_b)^3 X_2^3] \\
&\quad + \frac{1}{2}f^{(2)}(X_0 + \lambda_0 X_1)[(\lambda_c \lambda_d)(\lambda_e)^2 X_3^2]
\end{aligned} \tag{5.11}$$

(ii) Replacing $\lambda_0 X_1$ with $\lambda_0 \delta_{x1}$ in (5.11), where $\delta_{x1} = \frac{1}{2}(1 - 2^{-n_1})$ is the mid-value between the minimum and maximum values of X_1 , the iATA approximation function $P_A(X)$ without CDF is obtained as:

$$P_A(X) = D_0^A + D_1^A + D_2^A + D_3^A + D_4^A + D_5^A + D_6^A \tag{5.12}$$

where D_0^A , D_1^A and D_2^A are given earlier in (5.5), (5.6) and (5.8) respectively, and D_3^A , D_4^A , D_5^A and D_6^A given below are the error correction terms from (5.11).

$$\begin{aligned}
D_3^A &= +\frac{1}{2}f^{(2)}(X_0 + \lambda_0 \delta_{x1})[(\lambda_0 \lambda_1)^2 X_2^2] \\
D_4^A &= -\frac{1}{6}f^{(3)}(X_0 + \lambda_0 \delta_{x1})[(\lambda_a)(\lambda_b)^3 X_2^3] \\
D_5^A &= +\frac{1}{2}f^{(2)}(X_0 + \lambda_0 \delta_{x1})[(\lambda_a)(\lambda_b)^2 X_2^2] \\
D_6^A &= +\frac{1}{2}f^{(2)}(X_0 + \lambda_0 \delta_{x1})[(\lambda_c \lambda_d)(\lambda_e)^2 X_3^2]
\end{aligned}$$

5.2.2 Carry Prevention / Select Technique

In the approximation function $P_A(X)$, the two subtractions $(X_0 + \lambda_0 X_1) - \lambda_b X_2$ and $(X_0 + \lambda_0 X_1) - \lambda_e X_3$ in D_1^A (5.6) and D_2^A (5.8) respectively could result in a carry generated at the MSB position. Including the carry bit as part of the input to the tables implementing D_1^A and D_2^A will lead to larger tables. The carry problem and the two possible solutions are discussed below.

The carry problem is illustrated here with $n = 16$, $n_i = 4$ and $i = \{0, 1, 2, 3, a, b, c, d, e\}$ as a simplified example applying to D_1^A . In general, the problem exists in both D_1^A and D_2^A with any values of n and n_i . Let $x_{j,-k}$ be a bit in X_j ($j = \{0, 1, 2, 3\}$) at the $(-k)^{th}$ position of the input X and in_n the input to the table that implements D_1^A (5.6), i.e.

$$in_n = X_0 + \lambda_0 X_1 - \lambda_b X_2 \quad (5.13)$$

$$\begin{aligned}
&= \overbrace{x_{0,-1} x_{0,-2} x_{0,-3} x_{0,-4}}^{x_0} \overbrace{x_{1,-5} x_{1,-6} x_{1,-7} x_{1,-8}}^{\lambda_0 x_1} - \\
&\quad 0 \quad 0 \quad 0 \quad 0 \quad \overbrace{x_{2,-9} x_{2,-10} x_{2,-11} x_{2,-12}}^{\lambda_b x_2} \quad (5.14)
\end{aligned}$$

$$= c_a s_{-1} s_{-2} s_{-3} s_{-4} s_{-5} s_{-6} s_{-7} s_{-8} \quad (5.15)$$

where $x_{1-5}x_{1-6}x_{1-7}x_{1-8}$ are the 4 bits of X_1 shifted right by $n_0 = 4$ bits ($\lambda_0 = 2^{-n_0}$), $x_{2-9}x_{2-10}x_{2-11}x_{2-12}$ the 4 bits of X_2 shifted right by 4 bits too and c_a is the carry bit. There are n_b (4 in this case) leading zeros on the left of the bit x_{2-9} . The subtraction in (5.14) could generate a carry ($c_a = 1$) when $X_0 + \lambda_0 X_1 (= [0, 1 - 2^{-(n_0+n_1)}])$ is smaller than $\lambda_b X_2$, i.e. $X_0 = 0$ and $\lambda_0 X_1 < \lambda_b X_2$. One instance is $X_0 + \lambda_0 X_1 = "00000111"$ and $\lambda_b X_2 = "1000"$. As a result, the table would be twice the size as it has an extra bit in its input should the carry problem not be solved. Two possible solutions are investigated, namely the carry prevention technique (CPT) and the carry select technique (CST), for avoiding the extra carry bit as discussed below.

5.2.2.1 Carry Prevention Technique (CPT)

The CPT replaces in_n (5.13) with in_p (5.16) when all the bits of X_0 are ‘0’. In other words, the CPT prevents the carry generation by performing the subtraction in_n when there is a ‘1’ among the bits $x_{0,-1}$ to $x_{0,-4}$. Otherwise the addition in_p is performed. In this way, either in_n or in_p is performed and the carry at the MSB position is prevented, i.e., c_a is always zero.

$$in_p = X_0 + \lambda_0 X_1 + \lambda_b X_2 \quad (5.16)$$

Note that the CPT can be applied only if the central difference formulation is not employed because in the central difference formulation, both the subtraction and addition ((2.28) and (2.32)) are needed and the carry is inevitable.

When in_p is used as the input, the approximation equation is slightly different from $P_A(X)$ (5.12) where in_n is used. When using in_p , the approximation equation can be derived in the same way as in Section 5.2.1. However, the two terms $(\lambda_a + \lambda_c \lambda_d)f(X_0 + \lambda_0 X_1)$ in D_0^A (5.5) are replaced by $(-\lambda_a - \lambda_c \lambda_d)f(X_0 + \lambda_0 X_1)$. The resultant formulation contains exactly the same terms D_0^A , D_1^A , D_2^A , D_3^A , D_4^A , D_5^A and D_6^A but with different sign in some of the terms. Hence, the two approximation equations (one for in_p and the other for in_n) can be combined as given in (5.17) with s defined in (5.18) as the sign control. $P_B(X)$ is the same as $P_A(X)$ when $s = 1$ (in_n is applied). When $s = 0$, $P_B(X)$ is the approximation equation when in_p is used.

$$P_B(X) = D_0^B + D_1^B + D_2^B + D_3^B + D_4^B + D_5^B + D_6^B \quad (5.17)$$

where

$$\begin{aligned} D_0^B &= (1 - (-1)^s \lambda_a - (-1)^s \lambda_c \lambda_d) f(X_0 + \lambda_0 X_1) \\ D_1^B &= +(-1)^s \lambda_a f(X_0 + \lambda_0 X_1 + (-1)^s \lambda_b X_2) \\ D_2^B &= +(-1)^s \lambda_c \lambda_d f(X_0 + \lambda_0 X_1 + (-1)^s \lambda_e X_3) \\ D_3^B &= +\frac{1}{2} f^{(2)}(X_0 + \lambda_0 \delta_{x1}) [(\lambda_0 \lambda_1)^2 X_2^2] \\ D_4^B &= -\frac{1}{6} f^{(3)}(X_0 + \lambda_0 \delta_{x1}) [(\lambda_a)(\lambda_b)^3 X_2^3] \\ D_5^B &= -(-1)^s \frac{1}{2} f^{(2)}(X_0 + \lambda_0 \delta_{x1}) [(\lambda_a)(\lambda_b)^2 X_2^2] \\ D_6^B &= -(-1)^s \frac{1}{2} f^{(2)}(X_0 + \lambda_0 \delta_{x1}) [(\lambda_c \lambda_d)(\lambda_e)^2 X_3^2] \\ s &= \begin{cases} 0 & \text{all bits } x_{0,-1} \text{ to } x_{l,-\min(n_b, n_e)} \text{ are zero} \\ 1 & \text{Otherwise} \end{cases} \\ l &= \begin{cases} 1 & \min(n_b, n_e) > n_0 \\ 0 & \text{Otherwise} \end{cases} \end{aligned} \quad (5.18)$$

In general, when applying the CPT, the bits to be tested are $x_{0,-1}$ to $x_{l,-\min(n_b, n_e)}$. If $\min(n_b, n_e) > n_0$, the last bit to be tested is $x_{1,-\min(n_b, n_e)}$ ($l = 1$) instead of $x_{0,-\min(n_b, n_e)}$ ($l = 0$). Note that the test could be just on the bit $x_{0,-1}$ alone. However, more bits are tested in (5.18) to favor $s = 1$. $P_B(X)$ with $s = 1$ contains terms in D_1^A and D_2^A with alternate signs and gives smaller errors when higher order terms are truncated, resulting in smaller tables. This had been verified in the experiments.

5.2.2.2 Carry Select Technique (CST)

In the CST, the approximation equation $P_A(X)$ (5.12) is maintained and the carry may be generated. The term D_1^A (5.6) of $P_A(X)$ is implemented with two tables instead of a single one. The tables are then selected by the carry bit as described below.

The example in (5.13) is used again for illustration. The first table stores the $2^{n_0+n_1}$ entries corresponding to the cases that the carry bit $c_a = '0'$. In these cases, only bits s_{-1} to s_{-8} (5.15) but not c_a are used as input to the table. On the other hand, the second table stores the 2^{n_2} entries corresponding to the cases that $c_a = '1'$. Since the carry is generated ($c_a = '1'$) only when $X_0 = 0$, the bits s_{-1} to s_{-4} are all '1' in these cases. Therefore, the second table requires only the bits s_{-5} to s_{-8} as its inputs. In general, the first table stores $2^{\max(n_0+n_1, n_b+n_2)}$ entries while the second table stores 2^{n_2} entries. When $c_a = 0$, the first table output is selected whereas the second table output is selected when $c_a = 1$. The CST can be similarly applied to D_2^A (5.8).

It is worth to note that when applying the CST in iATA, either in_p or in_n can be used to derive the approximation formulation. However, in_n that leads to $P_B(X)$ with $s = 1$ (i.e. $P_A(X)$) is preferred to in_p (leads to $P_B(X)$ with $s = 0$) for the same reason discussed in the last paragraph of Section 5.2.2.1.

Both the CPT and the CST can reduce the table size. If the CPT is applied, only one table of the size same as the first table in the CST is needed, but more complex control logic is required. Therefore, the CPT is more advantageous in terms of table size with a saving of a table of size 2^{n_2} . However, the CPT will be no longer advantageous when it is integrated with the symmetry property exploitation (SPE) discussed in the next subsection. In the SPE, an additional constant term has to be added to in_n (5.13) and the carry cannot be prevented. Consequently, even if the CPT is applied, an additional table of size similar to the second table in the CST is still needed. On the other hand, the integration of the SPE with the CST does not incur any additional hardware overhead as there is still just one carry bit. In the current implementation, the CST was adopted for less control logic. Therefore, the approximation equation for iATA with the CST integrated remains as $P_A(X)$ (5.12).

5.2.3 Symmetry Property Exploitation (SPE)

Inspecting the formulation $P_A(X)$ in (5.12), it is observed that D_0^A , D_1^A and D_2^A do not possess the symmetrical property but D_3^A , D_4^A , D_5^A and D_6^A implicitly do. To exploit the property, X_2 and X_3 in D_3^A , D_4^A , D_5^A and D_6^A have to be replaced by $(X_2 - \delta_{x2})$ and $(X_3 - \delta_{x3})$ respectively, where $\delta_{x2} =$

$\frac{1}{2}(1 - 2^{-n_2})$ and $\delta_{x3} = \frac{1}{2}(1 - 2^{-n_3})$ are the mid-values between the minimum and maximum values of X_2 and X_3 . However, the replacement is not straightforward. The expressions of D_0^A , D_1^A and D_2^A have to be maintained after the replacement and preserving them is challenging as explained below. D_1^A and D_2^A are used to approximate the first derivatives of $f(X)$ without the CDF and they contain X_2 and X_3 . When X_2 and X_3 are substituted, the expressions of D_1^A and D_2^A will be altered. Consequently, the alterations in D_1^A and D_2^A will cause the expression of D_0^A to change too, resulting in inaccurate approximation. Therefore, some extra terms need to be added to D_0^A , D_1^A and D_2^A to maintain the accuracy of the approximation. With proper mathematical manipulations, the formulation for the SPE to be integrated is derived as follows.

Substituting X_2 and X_3 with $(X_2 - \delta_{x2})$ and $(X_3 - \delta_{x3})$ respectively, the function $f(X)$ is first rewritten as (5.19) where the extra constants $\delta_{2.3} = (\lambda_0\lambda_1\delta_{x2} + \lambda_0\lambda_1\lambda_2\delta_{x3})$ keep the function unaltered. Applying Taylor approximation with $X_m = (X_0 + \lambda_0X_1 + \delta_{2.3})$ as the new center, (5.22) is obtained.

The first derivative in (5.20) is approximated by D_1^C in (5.24) without the CDF. To facilitate the SPE, two extra constant terms $u = 2\lambda_b\delta_{x2} - \lambda_b\delta_{x2}$ ($= \lambda_b\delta_{x2}$) and $\delta_{2.3} = \lambda_0\lambda_1\delta_{x2} + \lambda_0\lambda_1\lambda_2\delta_{x3}$ are introduced in D_1^C . With u added to $-\lambda_bX_2$ in (5.24), $-\lambda_b(X_2 - \delta_{x2})$ is obtained. After Taylor approximation is applied to (5.24) and (5.26) is obtained, it becomes obvious that the symmetrical property can be exploited as the derivatives in (5.26) now contain $(X_2 - \delta_{x2})$ (symmetry to X_2). On the other hand, $\delta_{2.3}$ (part of X_m) is needed as it appears in the first derivative (5.20) that D_1^C is to approximate. In the Taylor expansion of D_1^C , the undesired term, $-\lambda_a f(X_m)$ (5.25), is cancelled out by its counterpart in D_0^C (5.23). Note that the computation of the added constant terms $-\lambda_b\delta_{x2} + \delta_{2.3}$ in (5.24) does not incur hardware cost because they can be pre-computed and merged with the function values in the tables. However, the term $2\lambda_b\delta_{x2}$ requires a simple inverter. Note that should the introduced term $u = 2\lambda_b\delta_{x2} - \lambda_b\delta_{x2}$ (for obtaining $-\lambda_b(X_2 - \delta_{x2})$) be simplified to $\lambda_b\delta_{x2}$, the circuit required would have been more complex as discussed in Section 5.4.

Similarly, D_2^C approximates the other first derivative (5.21) of the Taylor expansion of $f(X)$ with extra constant terms added in a similar way and for the similar reasons. Equation (5.27) can be derived in a similar way and its derivation is omitted for simplicity.

$$\begin{aligned} f(X) &= f(X_0 + \lambda_0X_1 + \lambda_0\lambda_1X_2 + \lambda_0\lambda_1\lambda_2X_3) \\ &= f(X_0 + \lambda_0X_1 + \lambda_0\lambda_1\delta_{x2} + \lambda_0\lambda_1\lambda_2\delta_{x3}) \end{aligned}$$

$$+\lambda_0\lambda_1(X_2 - \delta_{x2}) + \lambda_0\lambda_1\lambda_2(X_3 - \delta_{x3})) \quad (5.19)$$

$$= f(X_m) + f^{(1)}(X_m)[(\lambda_0\lambda_1)(X_2 - \delta_{x2})] + \quad (5.20)$$

$$f^{(1)}(X_m)[(\lambda_0\lambda_1\lambda_2)(X_3 - \delta_{x3})] + \quad (5.21)$$

$$\frac{1}{2}f^{(2)}(X_m)[(\lambda_0\lambda_1)^2(X_2 - \delta_{x2})^2 + 2(\lambda_0\lambda_1)^2(\lambda_2)(X_2 - \delta_{x2})(X_3 - \delta_{x3}) + (\lambda_0\lambda_1\lambda_2)^2(X_3 - \delta_{x3})^2] + \dots \quad (5.22)$$

$$D_0^C = (1 + \lambda_a + \lambda_c\lambda_d)f(X_m) \quad (5.23)$$

$$D_1^C = -\lambda_af(X_0 + \lambda_0X_1 - \lambda_bX_2 + 2\lambda_b\delta_{x2} - \lambda_b\delta_{x2} + \delta_{2.3}) \quad (5.24)$$

$$\begin{aligned} &= -\lambda_af(X_0 + \lambda_0X_1 + \delta_{2.3}) \\ &\quad -\lambda_af^{(1)}(X_m)\{(\lambda_b)[-X_2 + 2\delta_{x2} - \delta_{x2}]\} \\ &\quad -\frac{1}{2}\lambda_af^{(2)}(X_m)\{(\lambda_b)^2[-X_2 + 2\delta_{x2} - \delta_{x2}]^2\} \\ &\quad -\frac{1}{6}\lambda_af^{(3)}(X_m)\{(\lambda_b)^3[-X_2 + 2\delta_{x2} - \delta_{x2}]^3\} \\ &\quad -\frac{1}{24}\lambda_af^{(4)}(X_m)\{(\lambda_b)^4[-X_2 + 2\delta_{x2} - \delta_{x2}]^4\} + \dots \\ &= -\lambda_af(X_m) \end{aligned} \quad (5.25)$$

$$\begin{aligned} &+ \lambda_af^{(1)}(X_m)[(\lambda_b)(X_2 - \delta_{x2})] \\ &\quad -\frac{1}{2}\lambda_af^{(2)}(X_m)[(\lambda_b)^2(X_2 - \delta_{x2})^2] \\ &\quad +\frac{1}{6}\lambda_af^{(3)}(X_m)[(\lambda_b)^3(X_2 - \delta_{x2})^3] \\ &\quad -\frac{1}{24}\lambda_af^{(4)}(X_m)[(\lambda_b)^4(X_2 - \delta_{x2})^4] + \dots \end{aligned} \quad (5.26)$$

$$D_2^C = -\lambda_c\lambda_df(X_0 + \lambda_0X_1 - \lambda_eX_3 + 2\lambda_e\delta_{x3} - \lambda_e\delta_{x3} + \delta_{2.3}) \quad (5.27)$$

where

$$\begin{aligned} \delta_{2.3} &= \lambda_0\lambda_1\delta_{x2} + \lambda_0\lambda_1\lambda_2\delta_{x3} \\ X_m &= X_0 + \lambda_0X_1 + \delta_{2.3} \\ &= X_0 + \lambda_0X_1 + \lambda_0\lambda_1\delta_{x2} + \lambda_0\lambda_1\lambda_2\delta_{x3}. \end{aligned} \quad (5.28)$$

By repeating again the three-step derivations in Section 5.2.1 for D_0^C , D_1^C and D_2^C , the approximation equation $P_C(X)$ of the proposed iATA with SPE integrated is obtained as below. $P_C(X)$ is also the final formulation of the proposed iATA method.

$$P_{\text{iATA}}(X) = P_C(X) = D_0^C + D_1^C + D_2^C + D_3^C + D_4^C + D_5^C + D_6^C \quad (5.29)$$

where

$$D_3^C = +\frac{1}{2}f^{(2)}(X_p)[(\lambda_0\lambda_1)^2(X_2 - \delta_{x2})^2] \quad (5.30)$$

$$D_4^C = -\frac{1}{6}f^{(3)}(X_p)[(\lambda_a)(\lambda_b)^3(X_2 - \delta_{x2})^3] \quad (5.31)$$

$$D_5^C = +\frac{1}{2}f^{(2)}(X_p)[(\lambda_a)(\lambda_b)^2(X_2 - \delta_{x2})^2] \quad (5.32)$$

$$D_6^C = +\frac{1}{2}f^{(2)}(X_p)[(\lambda_c\lambda_d)(\lambda_e)^2(X_3 - \delta_{x3})^2] \quad (5.33)$$

and

$$\begin{aligned} X_p &= X_0 + \lambda_0\delta_{x1} + \delta_{2.3} \\ &= X_0 + \lambda_0\delta_{x1} + \lambda_0\lambda_1\delta_{x2} + \lambda_0\lambda_1\lambda_2\delta_{x3}. \end{aligned}$$

It is now apparent that the symmetrical property exists in the expressions of all the second and higher derivatives (5.30) to (5.33). Take D_6^C as an example. When the input to Table a_5 that implements D_6^C is $(2\delta_{x3} - X_3)$, which is the 1's complement of X_3 , the output value is the same as when the input is X_3 as illustrated below.

$$\begin{aligned} a_5(X_0, (2\delta_{x3} - X_3)) &= \frac{1}{2}f^{(2)}(X_p)[(\lambda_c\lambda_d)(\lambda_e)^2(2\delta_{x3} - X_3 - \delta_{x3})^2] \\ &= \frac{1}{2}f^{(2)}(X_p)[(\lambda_c\lambda_d)(\lambda_e)^2(X_3 - \delta_{x3})^2] \\ &= a_5(X_0, X_3) \end{aligned} \quad (5.34)$$

5.2.4 Final Formulation with Unequal Partitioning of Input

With different λ s applied to different parts of the input X (Fig. 2-5(d)), iATA partitions X into four unequal parts with wordlengths depending on the parameters λ s. Different choices of the parameters result in different table size and the parameter selection is constrained by the error budget of the function to be computed as discussed in the error analysis section below. The parameter selection will not, however, change the formulation $P_{iATA}(X)$ in (5.29). The terms of the final formulation $P_{iATA}(X)$ are implemented in tables and the outputs of the tables are then summed to obtain the value of the function to be evaluated. The architecture that implements $P_{iATA}(X)$ is depicted in Fig. 5-1 and described in Section 5.4, where the implementation issues of storing the terms in tables are also discussed.

5.3 Error Analysis

The final formulation $P_{iATA}(X)$ (5.29) contains several errors, including the error due to approximation and various rounding errors. By selecting appropriate values for the parameters (λ s), the errors can be controlled such that the evaluated function has faithful rounding result, i.e.

an error of at most one unit in the last place (ulp). The error analysis below facilitates the parameter selection and, hence the wordlength determination.

The total error $\varepsilon_{total} = \varepsilon_a + \varepsilon_r$ of $P_{iATA}(X)$ consists of two types of errors, the approximation error ε_a and the rounding error ε_r . The upper bound of ε_{total} is obtained with the two assumptions in (5.38) and (5.39). The details of the approximation error ε_a and rounding error ε_r are explained below.

5.3.1 Approximation Error ε_a

The approximation error ε_a consists of two errors, ε_{af} and ε_{ac} , as given in (5.35). ε_{af} (5.36) is the difference between the function $f(X)$ and its approximation $\tilde{P}_{iATA}(X)$. Here $\tilde{P}_{iATA}(X)$ is $P_{iATA}(X)$ in (5.29) before $\lambda_0 X_1$ is replaced by $\lambda_0 \delta_{x1}$ in the error correction terms. On the other hand, ε_{ac} (5.41) is an approximation error resulted from reducing part of the input ($X_0 + \lambda_0 X_1$) to the error correction terms in $\tilde{P}_{iATA}(X)$ to X_0 only, i.e. $\lambda_0 X_1$ is replaced by $\lambda_0 \delta_{x1}$ to obtain $P_{iATA}(X)$.

$$\varepsilon_a = \varepsilon_{af} + \varepsilon_{ac} \quad (5.35)$$

where

$$\begin{aligned} \varepsilon_{af} &= f(X) - \tilde{P}_{iATA} \\ &= f^{(1)}(X_m)[(\lambda_0 \lambda_1)(X_2 - \delta_{x2})] \\ &\quad - f^{(1)}(X_m)[(\lambda_a \lambda_b)(X_2 - \delta_{x2})] \\ &\quad + f^{(1)}(X_m)[(\lambda_0 \lambda_1 \lambda_2)(X_3 - \delta_{x3})] \\ &\quad - f^{(1)}(X_m)[(\lambda_c \lambda_d \lambda_e)(X_3 - \delta_{x3})] \\ &\quad + \frac{1}{2} f^{(2)}(X_m)[2(\lambda_0 \lambda_1)^2 (\lambda_2)(X_2 - \delta_{x2})(X_3 - \delta_{x3}) + (\lambda_0 \lambda_1 \lambda_2)^2 (X_3 - \delta_{x3})^2] \\ &\quad + \frac{1}{24} f^{(4)}(X_m)[(\lambda_a)(\lambda_b)^4 (X_2 - \delta_{x2})^4] \\ &\quad - \frac{1}{6} f^{(3)}(X_m)[(\lambda_c \lambda_d)(\lambda_e)^3 (X_3 - \delta_{x3})^3] + \dots, \end{aligned} \quad (5.37)$$

X_m is given earlier in (5.28),

$$n_0 + n_1 = n_a + n_b \Rightarrow \lambda_0 \lambda_1 = \lambda_a \lambda_b, \quad (5.38)$$

$$n_0 + n_1 + n_2 = n_c + n_d + n_e \Rightarrow \lambda_0 \lambda_1 \lambda_2 = \lambda_c \lambda_d \lambda_e \quad (5.39)$$

and ε_{ac} is given in (5.41).

The upper bound of ε_{af} given below is obtained by the Lagrange error bound [Phi96], which is the upper bound of Taylor series error. The equation of the Lagrange error bound and the detailed derivation of (5.40) are provided in Section 5.3.3.

$$\begin{aligned}\varepsilon_{af} &< 2^{-1} |f^{(2)}(\xi_2)| [2(\lambda_0\lambda_1)^2(\lambda_2)2^{-1}2^{-1} + (\lambda_0\lambda_1\lambda_2)^2(2^{-1})^2] \\ &\quad + \frac{1}{2^4} |f^{(4)}(\xi_4)| [(\lambda_a)(\lambda_b)^4(2^{-1})^4] \\ &\quad + \frac{1}{6} |f^{(3)}(\xi_3)| [(\lambda_c\lambda_d)(\lambda_e)^3(2^{-1})^3]\end{aligned}\tag{5.40}$$

where ξ_i , for $i = \{2,3,4\}$, is a point within the domain at which the derivative $f^{(i)}(X_m)$ is the maximum.

On the other hand, ε_{ac} given in (5.41) is introduced by the four correction terms D_3^C , D_4^C , D_5^C and D_6^C (5.30)–(5.33). The derivation of the most complex term ε_{D6} can be found in Section 5.3.3 while the derivations of the other three terms can be similarly obtained.

$$\varepsilon_{ac} = \varepsilon_{D3} + \varepsilon_{D4} + \varepsilon_{D5} + \varepsilon_{D6}\tag{5.41}$$

where

$$\varepsilon_{D3} < \frac{1}{2} [(\lambda_0\lambda_1)^2(2^{-1})^2] [|f^{(3)}(\xi_3)|(\lambda_0)2^{-1}]\tag{5.42}$$

$$\varepsilon_{D4} < \frac{1}{6} [(\lambda_a)(\lambda_b)^3(2^{-1})^3] [|f^{(4)}(\xi_4)|(\lambda_0)2^{-1}]\tag{5.43}$$

$$\varepsilon_{D5} < \frac{1}{2} [(\lambda_a)(\lambda_b)^2(2^{-1})^2] [|f^{(3)}(\xi_3)|(\lambda_0)2^{-1}]\tag{5.44}$$

$$\varepsilon_{D6} < \frac{1}{2} [(\lambda_c\lambda_d)(\lambda_e)^2(2^{-1})^2] [|f^{(3)}(\xi_3)|(\lambda_0)2^{-1}]\tag{5.45}$$

5.3.2 Rounding Error ε_r

The total rounding error ε_r given below includes a rounding error ε_{rt} contributed by the tables and an error due to rounding the final sum to pf fractional bits.

$$\varepsilon_r = \varepsilon_{rt} + \varepsilon_{rs} = \sum_{i=0}^{t-1} 2^{-pf-g_i-1} + 2^{-pf-1}$$

where $\varepsilon_{rt} = \sum_{i=0}^{t-1} 2^{-pf-g_i-1}$ is the total table rounding error contributed by t number of tables of a rounding error 2^{-pf-g_i-1} for table i . The table rounding errors are due to applying the rounding technique discussed later in Section 5.4 on the values stored in the tables with pf fractional bits (targeted precision) and g_i guard bits. And $\varepsilon_{rs} = 2^{-pf-1}$ is due to rounding the final sum of all the table outputs to pf fractional bits. In iATA, it is ensured that the total error ε_{total} is smaller than one ulp to fulfill the faithful rounding requirement, i.e. $\varepsilon_{total} < 2^{-pf}$.

Inspecting the error expressions of ε_{af} and ε_{ac} , it is noticed that the error is highly dependent on the third derivative of $f(X)$. The error expressions show that the total error can be restrained by choosing appropriate values for the design parameters n_i ($i = \{0,1,2,3,a,b,c,d,e\}$) that adhere to the condition $n = n_0 + n_1 + n_2 + n_3$, and the assumptions (5.38) and (5.39). The following set of design parameters is proposed for the binary logarithm function.

$$S_A = \{ n_0 = \frac{n}{4} + 1, n_1 = \frac{n}{4} - 1, n_2 = \frac{n}{4} - 1, n_3 = \frac{n}{4} + 1, \\ n_a = \frac{n}{4} - 1, n_b = \frac{n}{4} + 1, n_c = \frac{n}{4}, n_d = \frac{n}{4} - 1, n_e = \frac{n}{4} \}$$

where $n = 24$ is the wordlength of the input X . The value of n_3 is reduced by 1 as there is one constant integer bit in the input as shown in Table 5-1.

The error analysis also provides insights into why the SPE helps to restrain the errors in addition to the benefit of 50% table size reduction. Consider ε_{D6} as an example. Without the SPE, ε_{D6} would have been as follows:

$$\varepsilon_{D6} < \frac{1}{2} [(\lambda_c \lambda_d)(\lambda_e)^2 |X_3|^2] [|f^{(3)}(\xi_3)|(\lambda_0)|X_1 - \delta_{x1}|] \\ < \frac{1}{2} [(\lambda_c \lambda_d)(\lambda_e)^2 (1)^2] [|f^{(3)}(\xi_3)|(\lambda_0)2^{-1}] \quad (5.46)$$

Comparing (5.46) with (5.45), it is evident that the SPE reduces the maximum value of X_3 in ε_{D6} by half and hence, the maximum error by three quarters, i.e. from $(X_3)^2 < (1)^2$ in (5.46) to $(X_3 - \delta_{x3})^2 < (2^{-1})^2$ in (5.45). The reduction is significant because of the quadratic effect on X_3 . The benefit of SPE to error reduction can be witnessed similarly in the other error expressions (5.40), (5.42), (5.43) and (5.44).. Note that the benefit of SPE to error reduction is not mentioned in any of the previous methods (SBTM, STAM and MTM) that apply it.

5.3.3 Derivations

5.3.3.1 Upper Bound of ε_{af}

With the assumptions (5.38) and (5.39), ε_{af} in (5.37) is simplified to (5.48). Based on the Lagrange error bound [Phi96] given in (5.47), which is the upper bound of Taylor series error, the upper bound for ε_{af} is obtained in (5.49) and resulted in (5.50) as $|X_2 - \delta_{x2}| < 2^{-1}$ and $|X_3 - \delta_{x3}| < 2^{-1}$ (δ_{x2} and δ_{x3} are the respective mid-values of X_2 and X_3).

$$\varepsilon_{UB,Taylor} = \frac{f^{(n+1)}(\xi)(X-a)^{n+1}}{(n+1)!} \quad (5.47)$$

where a is where the series is centered and ξ is a value between a and X that gives the largest value for $f^{(n+1)}(\xi)$.

$$\begin{aligned}\varepsilon_{af} = & \frac{1}{2}f^{(2)}(X_m)[2(\lambda_0\lambda_1)^2(\lambda_2)(X_2 - \delta_{x2})(X_3 - \delta_{x3}) + (\lambda_0\lambda_1\lambda_2)^2(X_3 - \delta_{x3})^2] \\ & + \frac{1}{24}f^{(4)}(X_m)[(\lambda_a)(\lambda_b)^4(X_2 - \delta_{x2})^4] \\ & - \frac{1}{6}f^{(3)}(X_m)[(\lambda_c\lambda_d)(\lambda_e)^3(X_3 - \delta_{x3})^3] + \dots\end{aligned}\quad (5.48)$$

$$\begin{aligned}< \frac{1}{2}|f^{(2)}(\xi_2)|[2(\lambda_0\lambda_1)^2(\lambda_2)|X_2 - \delta_{x2}||X_3 - \delta_{x3}| + (\lambda_0\lambda_1\lambda_2)^2|X_3 - \delta_{x3}|^2] \\ & + \frac{1}{24}|f^{(4)}(\xi_4)|[(\lambda_a)(\lambda_b)^4|X_2 - \delta_{x2}|^4] \\ & + \frac{1}{6}|f^{(3)}(\xi_3)|[(\lambda_c\lambda_d)(\lambda_e)^3|X_3 - \delta_{x3}|^3]\end{aligned}\quad (5.49)$$

$$\begin{aligned}< \frac{1}{2}|f^{(2)}(\xi_2)|[2(\lambda_0\lambda_1)^2(\lambda_2)2^{-1}2^{-1} + (\lambda_0\lambda_1\lambda_2)^2(2^{-1})^2] \\ & + \frac{1}{24}|f^{(4)}(\xi_4)|[(\lambda_a)(\lambda_b)^4(2^{-1})^4] \\ & + \frac{1}{6}|f^{(3)}(\xi_3)|[(\lambda_c\lambda_d)(\lambda_e)^3(2^{-1})^3]\end{aligned}\quad (5.50)$$

where X_m is given in (5.28).

5.3.3.2 Upper Bound of ε_{D6}

Replacing $\lambda_0 X_1$ with $\lambda_0 \delta_{x1}$ in D_6^C (5.33) leads to approximation error ε_{D6} given below. Equation (5.52) is obtained by applying Taylor series to (5.51) while (5.54) is derived from applying the Lagrange error bound to (5.53), which results in (5.55).

$$\begin{aligned}\varepsilon_{D6} &= \frac{1}{2}[(\lambda_c\lambda_d)(\lambda_e)^2(X_3 - \delta_{x3})^2] \cdot \\ & \quad [f^{(2)}(X_m) - f^{(2)}(X_m - \lambda_0(X_1 - \delta_{x1}))]\end{aligned}\quad (5.51)$$

$$\begin{aligned}&= \frac{1}{2}[(\lambda_c\lambda_d)(\lambda_e)^2(X_3 - \delta_{x3})^2] \cdot \\ & \quad \{f^{(2)}(X_m) - [f^{(2)}(X_m) + f^{(3)}(X_m)(-\lambda_0)(X_1 - \delta_{x1}) + \\ & \quad \frac{1}{2}f^{(4)}(X_m)(-\lambda_0)^2(X_1 - \delta_{x1})^2 + \dots]\}\end{aligned}\quad (5.52)$$

$$\begin{aligned}&= \frac{1}{2}[(\lambda_c\lambda_d)(\lambda_e)^2(X_3 - \delta_{x3})^2] \cdot \\ & \quad [-f^{(3)}(X_m)(-\lambda_0)(X_1 - \delta_{x1}) - \\ & \quad \frac{1}{2}f^{(4)}(X_m)(-\lambda_0)^2(X_1 - \delta_{x1})^2 - \dots]\end{aligned}\quad (5.53)$$

$$< \frac{1}{2} [(\lambda_c \lambda_d)(\lambda_e)^2 |X_3 - \delta_{x3}|^2] [|f^{(3)}(\xi_3)|(\lambda_0)|X_1 - \delta_{x1}|] \quad (5.54)$$

$$< \frac{1}{2} [(\lambda_c \lambda_d)(\lambda_e)^2 (2^{-1})^2] [|f^{(3)}(\xi_3)|(\lambda_0)2^{-1}] \quad (5.55)$$

5.4 Hardware Architecture

The formulation $P_{iATA}(X)$ (5.29) of the proposed iATA method is realized by the architecture depicted in Fig. 5-1. The functions of the components are explained below.

5.4.1 Tables

The tables a_0, a_1, a_2, a_3, a_4 and a_5 implement $D_0^C, D_1^C, D_2^C, (D_3^C + D_5^C), D_4^C$ and D_6^C respectively. As discussed in Section 5.2.2.2, two tables are used to implement D_1^C when the CST is applied. Hence, a small table $a_{1.1}$ is used in addition to a_1 . $a_{1.1}$ is used when carry is generated in the input in_n (5.13). After the integration of SPE, the input in_n becomes in_1 given in (5.56), which is obtained from (5.24) where the constant $-\lambda_b \delta_{x2} + \delta_{2.3}$ is excluded from the input to table and the other constant $2\lambda_b \delta_{x2}$ is merged with $-\lambda_b X_2$ to give $\lambda_b \overline{X_2}$. Note that δ_{x2} is the mid-value of X_2 , $2\delta_{x2}$ is the maximum of X_2 and contains all 1's and hence $2\delta_{x2} - X_2$ is the 1's complement of X_2 , i.e. $\overline{X_2}$. The situation that the carry is generated in in_1 and the entries for table $a_{1.1}$ are described below. The example with $n = 16$ and $n_i = 4$ ($\lambda_i = 2^{-4}$) for $i = \{0, 1, 2, 3, a, b, c, d, e\}$ is used as an illustration here. The carry c_a is generated ($c_a = '1'$) when $X_0 = "1111"$ and $(\lambda_0 X_1 + \lambda_b \overline{X_2}) \geq 2^{-n_0}$. The example below is one instance.

$$\begin{aligned} in_1 &= X_0 + \lambda_0 X_1 - \lambda_b X_2 + 2\lambda_b \delta_{x2} \\ &= X_0 + \lambda_0 X_1 + \lambda_b \overline{X_2} \\ &= \overbrace{\begin{matrix} X_0 = x_{0,-1} & x_{0,-2} & x_{0,-3} & x_{0,-4} \end{matrix}}^{\lambda_0 X_1 = x_{1,-5} \quad x_{1,-6} \quad x_{1,-7} \quad x_{1,-8}} \begin{matrix} 1 & 1 & 1 & 1 \end{matrix} \quad \begin{matrix} 1 & 0 & 0 & 0 \end{matrix} + \\ &\quad \begin{matrix} 0 & 0 & 0 & 0 \end{matrix} \quad \overbrace{\begin{matrix} \lambda_b \overline{X_2} = x_{2,-9} & x_{2,-10} & x_{2,-11} & x_{2,-12} \end{matrix}}^{\lambda_b \overline{X_2} = x_{2,-9} \quad x_{2,-10} \quad x_{2,-11} \quad x_{2,-12}} \begin{matrix} 1 & 0 & 0 & 0 \end{matrix} \\ &= \begin{matrix} c_a & s_{-1} & s_{-2} & s_{-3} & s_{-4} & s_{-5} & s_{-6} & s_{-7} & s_{-8} \end{matrix} \end{aligned} \quad (5.56)$$

Fig. 5-1: iATA architecture

In the case that the carry c_a is generated, $s_{-1}s_{-2}s_{-3}s_{-4}$ will be all 0's. Table $a_{1.1}$ will store the values of $D_{1.1}^C$ (5.57) and its output is selected by c_a . Equation (5.57) is exactly the same as D_1^C (5.24) except that it is computed for only the entries when $s_{-1}s_{-2}s_{-3}s_{-4} = "0000"$ and $c_a = '1'$. In a similar way, Table $a_{2.1}$, storing $D_{2.1}^C$ (5.58), is added in parallel to Table a_2 to completely realize D_2^C . Therefore, a total of 8 tables are used in the iATA architecture.

$$D_{1.1}^C = -\lambda_a f(1. \overbrace{00 \dots 0}^{n_b \text{ bits}} + in_{11} - \lambda_b \delta_{x_2} + \delta_{2.3}) \quad (5.57)$$

where

$$in_{11} = n_2 \text{ LSBs of } (X_0 + \lambda_0 X_1 + \lambda_b \overline{X_2})$$

$$D_{2.1}^C = -\lambda_c \lambda_d f(1. \overbrace{00 \dots 0}^{n_e \text{ bits}} + in_{21} - \lambda_e \delta_{x_3} + \delta_{2.3}) \quad (5.58)$$

where

$$in_{21} = n_3 \text{ LSBs of } (X_0 + \lambda_0 X_1 + \lambda_e \overline{X_3}) \quad (5.59)$$

5.4.2 NOT and XOR Gates

The NOT and XOR gates are all for logical negations. NOT₁ realizes $\lambda_b \overline{X_2}$ of (5.56). It can be seen that should the term $2\lambda_b \delta_{x_2}$ be omitted, the ordinary 2's complement would have to be performed for $-\lambda_b X_2$, resulting in more complex hardware. The same applies to NOT₂, which realizes $-\lambda_e X_3 + 2\lambda_e \delta_{x_3} = \lambda_e \overline{X_3}$ of in_{21} in (5.59).

On the other hand, XOR_1 , XOR_2 and XOR_3 are required for 1's complement as the result of exploiting the symmetrical property (5.34) of the tables a_3 , a_4 and a_5 . Note that negation is not needed at the outputs of Tables a_3 and a_5 because $(X_2 - \delta_{x2})$ and $(X_3 - \delta_{x3})$ are both squared in (5.30), (5.32) and (5.33). The signals $\tilde{X}_{2,3}$ (and $\tilde{X}_{2,4}$) and $\tilde{X}_{3,5}$ in Fig. 5-1 denote respectively X_2 and X_3 with some of their least significant bits (LSBs) truncated without violating the accuracy requirement. The number of bits truncated varies from function to function and is determined empirically as follows. Given a function, the formulation $P_{iATA}(X)$ (5.29) is simulated in Matlab without truncation for the design parameter set S_A to verify the accuracy. Subsequently, the simulation is repeated with increasing number of truncated LSBs for $\tilde{X}_{2,3}$, $\tilde{X}_{2,4}$ and $\tilde{X}_{3,5}$ until the accuracy is violated. The best results in terms of total table size are then implemented (see Table 5-5).

It is worth to mention that the arithmetic negation of Table a_4 output in iATA is realized by simple logical negation instead of the conventional 2's complement. This is possible because the rounding technique proposed in MTM (Section 4.6.2 of [deD05]) is adopted. The technique assumes a '1' at the bit position $pf + g_i + 1$ of Table i , for $i = \{0, 1, 2, 3, 4, 5\}$ and retains the '1' bit after negation. As such, the usual addition of '1' during 2's complement is avoided by adding the value $\sum_{i=0}^5 2^{-(pf+g_i+1)}$ (for 6 tables) into one of the tables. The rounding technique is also applied at the final output and an additional value of $2^{-(pf+1)}$ is added to the table too.

The iATA architecture in Fig. 5-1 is modeled, simulated and implemented as described in Section 5.5 and 5.6.

5.5 Performance Evaluations for Binary Logarithm Function

5.5.1 Table Size

The iATA architecture in Fig. 5-1 was modeled and simulated in Matlab for the binary logarithm function, $\log_2(X)$, with its corresponding input domain (Dom) and number of integer (I) and fractional (F) bits of the input and output as listed in Table 5-1. The binary logarithm function $\log_2(X)$ has input domain $[1, 2)$. Thus, the MSB of X is always '1' and is not part of the input. Exhaustive simulation is performed to verify that the evaluated values of the functions are

TABLE 5-1: IMPLEMENTATION DETAILS FOR $\log_2(X)$ GENERATED BY IATA

Word-length, n	Dom	Input		Output		Trunc bits			Number of guard bits					
		I	F	I	F, pf	$\tilde{X}_{2,3}$	$\tilde{X}_{2,4}$	$\tilde{X}_{3,5}$	g_0	g_1/g_{11}	g_2/g_{21}	g_3	g_4	g_5
20 bits	[1, 2)	1	$n-1$	0	n	0	0	0	2	2	2	3	3	3
24 bits						0	0	3	2	3	3	3	3	3

accurate up to one ulp with the IEEE double precision results from the corresponding Matlab functions as references. The number of truncated bits (Trunc bits) for $\tilde{X}_{2,3}$, $\tilde{X}_{2,4}$ and $\tilde{X}_{3,5}$, and number of guard bits (g_i) for the tables are included in Table 5-1. As mentioned in Section 1.2, the iATA aims to compute binary logarithm for high accuracy requirement (>18 bits) beyond those that can be achieved by the Mitchell-based methods, thus, the iATA architecture in Fig. 5-1 is simulated and implemented for $n = 20$ and 24 bits input/output (IO) wordlength. 24 bits IO is the maximum IO wordlength for which there are results reported by the existing tables-and-additions methods. 20 bits IO is a wordlength conveniently chosen between 18 and 24 bits. In general, the tables-and-additions methods can be applied, with certain effort, to other IO wordlengths.

Table 5-2 tabulates the size of each table, $2^{wi_i} \times wo_i$, where wi_i and wo_i are respectively the input and output wordlengths of the table a_i for $i = \{0,1,1.1,2,2.1,3,4,5\}$. The size is calculated based on only wi_i and wo_i as in the conventional way used by the existing methods for comparisons. wo_i varies from function to function because leading zero/one bits and a constant sign bit are removed. $wo_4 = 0$ indicates that a_4 is not required as all its entries are too small and negligible

In Table 5-3, the table size for computing the function $\log_2(X)$ by iATA is compared with the existing results by SBTM [Sch99] of the tables-and-additions category as other methods in the same category does not report the results for $\log_2(X)$. As shown in Table 5-3, the table sizes required by the proposed iATA method are 75.31% (or more than four times smaller) and 87.22% (or more than seven times smaller) smaller compared to that of SBTM for $n = 20$ and 24 bits input/output respectively.

TABLE 5-2: TABLE SIZES FOR $\log_2(X)$ GENERATED BY iATA

n (bits)	Table size (bits)								Total
	a_0	a_1	$a_{1.1}$	a_2	$a_{2.1}$	a_3	a_4	a_5	
20	$2^{10} \times 23$	$2^{10} \times 19$	$2^4 \times 19$	$2^{10} \times 14$	$2^5 \times 14$	$2^9 \times 5$	$2^9 \times 0$	$2^{10} \times 2$	62704
24	$2^{12} \times 27$	$2^{12} \times 23$	$2^5 \times 23$	$2^{12} \times 17$	$2^6 \times 17$	$2^{11} \times 6$	$2^{11} \times 0$	$2^9 \times 2$	289568

TABLE 5-3: TABLE SIZE (BITS) COMPARISONS FOR $\log_2(X)$

n (bits)	SBTM [Sch99]	iATA	Improvement (%)
20	253,952	62,704	75.31
24	2,031,616	289,568	87.22

5.5.2 Standard Cell Implementation

The iATA architecture in Fig. 5-1 can be realized in any technology, including standard-cell based integrated circuits (ICs) and FPGA devices. As the logarithmic converters obtained by the proposed UMA and RM in Chapter 3 and 4 are synthesized in STM90nm CMOS technology, the iATA architecture for computing $\log_2(X)$ with $n = 20$ and 24 bits input/output is coded in Verilog HDL and synthesized in the same technology for better analysis of the trade-offs between accuracy and hardware merits as discussed in Section 6.1. The synthesis results are tabulated in Table 5-4. Note that no synthesis result is reported in SBTM [Sch99] for comparison and its design cannot be synthesized as the table contents are not available.

5.6 Performance Evaluations for Other Elementary Functions

5.6.1 Table Size

The iATA architecture in Fig. 5-1 was also modeled and simulated in Matlab for the elementary functions listed in Table 5-5 with their corresponding input domains (Dom) and number of integer (I) and fractional (F) bits of the input and output. The elementary functions in Table 5-5 are considered because they are used for performance comparisons by STAM [Sti99] and/or MTM [deD05] which are the two best tables-and-additions methods. 24 bits IO is considered for the same reason discussed in Section 5.5. The functions $\frac{1}{X}$, $\sqrt{X}$ and $\ln(X)$ have input domain $[1, 2)$. Thus, the MSB of X is always '1' and is not part of the input. Table 5-4, Table 5-5 and Table 5-6 in this section are similar to Table 5-1, Table 5-2 and Table 5-3, respectively, except that an additional column on the set of design parameters used for each elementary functions is added in Table 5-5. In other words, beside S_A , another set of design parameter, S_B , is proposed as follows.

TABLE 5-4: SYNTHESIS RESULTS FOR $\log_2(X)$

n (bits)	A (μm^2)	r (ns)	ADP
20	83,257	3.97	330,530
24	305,295	5.39	1,645,540

TABLE 5-5: IMPLEMENTATION DETAILS FOR FUNCTIONS GENERATED BY IATA WITH 24-BIT INPUT/OUTPUT

Func	Dom	Input		Output		Set	Trunc bits			Number of guard bits					
		I	F	I	F, pf		$\tilde{X}_{2,3}$	$\tilde{X}_{2,4}$	$\tilde{X}_{3,5}$	g_0	g_1/g_{11}	g_2/g_{21}	g_3	g_4	g_5
$\frac{1}{X}$	$[1, 2)$	1	$n-1$	0	n	S_A	0	3	3	2	3	3	3	3	3
2^X	$[0, 1)$	0	n	1	$n-1$	S_B	0	3	3	3	3	2	3	3	3
$\sin(X)$	$[0, \frac{\pi}{4})$	0	n	0	n	S_B	0	2	3	3	3	3	3	2	3
$\sqrt{X}$	$[1, 2)$	1	$n-1$	1	$n-1$	S_B	1	3	3	2	3	3	3	3	1
$\ln(X)$	$[1, 2)$	1	$n-1$	0	n	S_B	0	1	0	3	5	5	5	5	5

TABLE 5-6: TABLE SIZES FOR FUNCTIONS GENERATED BY IATA WITH 24-BIT INPUT/OUTPUT

Function	Table size (bits)								
	a_0	a_1	$a_{1,1}$	a_2	$a_{2,1}$	a_3	a_4	a_5	Total
$\frac{1}{x}$	$2^{12} \times 27$	$2^{12} \times 23$	$2^5 \times 22$	$2^{12} \times 17$	$2^6 \times 16$	$2^{11} \times 6$	$2^8 \times 1$	$2^9 \times 2$	289,728
2^x	$2^{11} \times 28$	$2^{11} \times 21$	$2^6 \times 22$	$2^{12} \times 14$	$2^7 \times 15$	$2^{11} \times 7$	$2^8 \times 0$	$2^9 \times 1$	175,872
$\sin(X)$	$2^{11} \times 27$	$2^{11} \times 22$	0	$2^{12} \times 16$	0	$2^{11} \times 8$	$2^9 \times 1$	$2^9 \times 2$	183,808
$\sqrt{x}$	$2^{11} \times 26$	$2^{11} \times 21$	$2^6 \times 21$	$2^{11} \times 15$	$2^6 \times 15$	$2^{10} \times 5$	$2^8 \times 0$	$2^8 \times 1$	134,656
$\ln(X)$	$2^{11} \times 27$	$2^{11} \times 24$	$2^6 \times 23$	$2^{11} \times 18$	$2^6 \times 17$	$2^{11} \times 10$	$2^{10} \times 4$	$2^{11} \times 5$	178,688

$$S_B = \{ n_0 = \frac{n}{4}, n_1 = \frac{n}{4} - 1, n_2 = \frac{n}{4}, n_3 = \frac{n}{4} + 1, \\ n_a = \frac{n}{4}, n_b = \frac{n}{4} - 1, n_c = \frac{n}{4}, n_d = \frac{n}{4}, n_e = \frac{n}{4} - 1 \}$$

The set S_B leads to smaller tables than S_A does. However, the functions implemented with S_B are required to have relatively small values of derivatives for meeting the accuracy requirement. In other words, S_A is more suitable for functions with large values of derivatives especially in the third derivative, such as the reciprocal function. Each function in Table 5-5 is first simulated with the set S_B . If the accuracy requirement is met, it is used. Otherwise, S_A is used. Similar to the case in S_A , for those functions with one constant integer bit in their inputs, the value of n_3 is reduced by 1. In this work, S_B was also simulated for the binary logarithm function but the accuracy requirement was not met. Table 5-6 tabulates the size of each table, $2^{wi_i} \times wo_i$, where wi_i and wo_i are respectively the input and output wordlengths of the table a_i for $i = \{0, 1, 1.1, 2, 2.1, 3, 4, 5\}$. wo_i varies from function to function because leading zero/one bits and a constant sign bit are removed. $wo_4 = 0$ indicates that a_4 is not required as all its entries are too small and negligible. Note that $a_{1,1}$ and $a_{2,1}$ are not required for $\sin(X)$ because the condition for the carry to be generated in in_1 and in_2 will never occur within its input domain.

In Table 5-7, the table size is compared for evaluating the five common elementary functions using iATA and the existing methods in the same (tables-and-additions) category. The memory required by ATA is the same for all the functions because no optimization is performed. The memory size for $\ln(X)$ is not reported by SBTM and MTM. For $\sqrt{x}$, SBTM reported the result for

TABLE 5-7: TABLE SIZE (BITS) COMPARISONS FOR $n = 24$

Function	ATA [Won95]	SBTM [Sch99]	Multipartite Methods		iATA	% Improvement by iATA over			
			STAM [Sti99]	MTM [deD05]		ATA	SBTM	STAM	MTM
$\frac{1}{x}$	868,352	1,933,312	651,264	379,392 [#]	289,728	66.63	85.01	55.51	23.63
2^x	868,352	1,474,560	356,352	207,872 [#]	175,872	79.75	88.07	50.65	15.39
$\sin(X)$	868,352	1,998,848	491,520	189,440 [#]	183,808	78.83	90.80	62.60	2.97
$\sqrt{x}$	868,352	NA	178,176 [#]	NA	134,656	84.49	-	24.43	-
$\ln(X)$	868,352	NA	462,248 [#]	NA	178,688	79.42	-	61.39	-

*Obtained from [42]; #Best existing results; NA: Not Available.

a different input domain of $[1, 4)$ while MTM did not report the result.

Among the existing methods, MTM is the most memory efficient method. The proposed iATA method achieves higher memory efficiency than MTM does with a memory saving of 23.63% for $\frac{1}{x}$. iATA also performs well for 2^x , achieving 15.39% saving, and is comparable to MTM for $\sin(X)$. For $\sqrt{x}$ and $\ln(X)$, iATA is substantially more efficient than the best existing results by STAM with respective memory saving of 24.43% and 61.39%.

5.6.2 FPGA Implementations

For comparison with the best existing design (MTM [deD05]), the iATA architecture in Fig. 5-1 is coded in Verilog HDL and implemented in FPGA devices, the same implementation option as the MTM [deD05] which is the best tables-and-additions method. The five functions listed in Table 5-8 are implemented first with an older and then a latest Xilinx FPGA device. The old device and tool are used for a fair comparison with MTM [deD05]. The same Optimization Goal (Area) and Effort (High) are also used during synthesis. MTM implemented three functions ($\frac{1}{x}$, 2^x and $\sin(X)$) but for 24-bit input, only $\sin(X)$ could be fitted into the device. For iATA, four functions can be fitted into the device. For $\sin(X)$, iATA requires 0.2% more slices than MTM does and has the same slice utilization of 97%. This is consistent with the total table size (Table 5-7) as they have a comparable total table size for $\sin(X)$. On the other hand, iATA achieves a shorter pad to pad delay with a reduction of 33.3%. The speed improvement was unexpected. The two architectures have similar structure and should have a comparable delay. One possible cause might be the HDL coding style for the 6-operand adder. In iATA, the additions are coded as $s_1 = a_0 + a_1$; $s_2 = a_2 + a_3$; $s_3 = a_4 + a_5$; $s_4 = s_1 + s_2$; $sum = s_3 + s_4$. When coded as $s_1 = a_0 + a_1$; $s_2 = a_2 + s_1$; $s_3 = a_3 + s_2$; $s_4 = a_4 + s_3$; $sum = a_5 + s_4$, a worse delay was reported by Xilinx ISE synthesizer.

TABLE 5-8: COMPARISONS OF HARDWARE IMPLEMENTATION IN VIRTEX II

Method	MTM [deD05]		iATA								
Tool	ISE 5.2.03i		ISE 5.2.03i						ISE 10.1		
Device	XC2V1000-fg456-5		XC2V1000-fg456-5			XC2V1500-fg676-5			XC2V1000-fg456-5		
Design	Non-pipelined		Non-pipelined			Non-pipelined			3-stage Pipelined		
Func	Slices 5120 (100%)	Delay (ns)	Slices 5120 (100%)	$\Delta\%$	Delay (ns)	$\Delta\%$	Slices 7680 (100%)	Delay (ns)	Slices 5120 (100%)	RAMB16 40 (100%)	Period (ns)
$\frac{1}{x}$	-	-	-	-	-	-	7454 (97%)	31.7	68 (1.3%)	22 (55%)	8.99
2^x	-	-	4795 (93%)		30.9	-	4795 (62%)	32.5	56 (1.1%)	14 (35%)	7.97
$\sin(X)$	4954 (97%)	43.0	4963 (97%)	-0.2	28.7	33.3	4963 (64%)	31.9	65 (1.3%)	12 (30%)	8.94
$\sqrt{x}$	-	-	3386 (66%)	-	30.0	-	3386 (44%)	30.9	59 (1.2%)	11 (27%)	8.69
$\ln(X)$	-	-	5020 (98%)	-	27.9	-	5020 (65%)	31.9	74 (1.4%)	14 (35%)	8.92

TABLE 5-9: IMPLEMENTATION RESULTS IN VIRTEX 7 WITH ISE 13.1

Device	Virtex 7 XC7V285t-ffg484-3				
Design	3-stage Pipelined			Non-pipelined	
Func	Slices	RAMB36E1/ 18E1	Period (ns)	Slices	Delay (ns)
$\frac{1}{x}$	37 (0.1%)	8 / 4 (2.4%)	4.79	1108 (2.5%)	14.95
2^x	34 (0.1%)	5 / 5 (1.8%)	3.91	664 (1.5%)	13.98
$\sin(X)$	34 (0.1%)	4 / 4 (1.5%)	3.81	727 (1.6%)	13.97
$\sqrt{X}$	34 (0.1%)	3 / 5 (1.3%)	3.83	487 (1.1%)	13.78
$\ln(X)$	45 (0.1%)	4 / 6 (1.7%)	4.88	683 (1.5%)	13.96

Investigation is also carried out on the size and delay for $\frac{1}{x}$ by implementing it in a larger device of the same family, which might not be available to MTM at its time. This function can be fitted into the device. The number of slices required by the other four functions remains the same, but the pad to pad delays increase due to longer routing delays in the larger device.

For designs implemented in FPGA devices, synchronous and pipelined structure is recommended [Xil05]. It is also recommended to use on-chip hardware resources to achieve high performance. As such, the iATA architecture was pipelined, the on-chip memory (BlockRAM) was used for the tables and its performance was investigated. The XC2V1000 device was first used for comparisons and then a device from the latest Virtex 7 family as discussed below.

The architecture is pipelined into three stages. The first stage consists of the adders before the tables and the tables. The second stage consists of the four additions s_1 , s_2 , s_3 and s_4 mentioned above. The third stage encompasses only the addition sum . The implementation results of XC2V1000 are included in Table 5-8. Noted that ISE 10.1, which is the latest version that supports Virtex II devices, was used because ISE 5.2.03i was not able to synthesize the tables into the block RAMs. The pipelined architecture can be clocked with a clock period less than 9ns. Note that in Table 5-8, the clock period of the 3-stage pipelined architecture is less than one third of the clock period of the non-pipelined architecture because high performance on-chip memory (BlockRAM) was used for the tables. The slice utilization is less than 1.4%, leaving resource for other logics. The utilization of 18k-Bit block RAMs ranges from 27% to 55%.

The 3-stage pipelined and the non-pipelined architectures were also implemented in a Virtex 7 device with the latest tool (ISE 13.1). The results are tabulated in Table 5-9. The pipelined designs can operate with a clock period of 4 to 5 ns. They occupy a small number of slices at the expense of memories. Note that the XC7V285t has 44700 slices and 410 RAMB36E1s (or 820 RAMB18E1s).

It is worth to note that designs implemented in FPGA devices usually adopt synchronous pipelined architecture. The evaluation of elementary functions such as the reciprocal is often a small task in a large application system such as 3-D computer graphics processing. It is desirable to keep memory resources consumed by these functions as small as possible, leaving resources for data. Table 5-9 shows that these functions occupy only a very small number of slices but a sizable portion of memory. Therefore, reducing memory used by the elementary functions is beneficial. Furthermore, when the elementary functions are implemented in standard-cell based ICs, smaller tables enable the ROM to be synthesized by industrial tool and reduce the overall chip size and cost.

5.7 Summary

A new tables-and-additions method for high accuracy computation of elementary functions is presented. The proposed method, named as iATA, formulates the approximation equation without using the central difference formulation. It integrates three techniques, namely the carry select technique, the symmetry property exploitation and the unequal partitioning of the input with the aid of error analysis. The detailed mathematical formulations were derived, the method was simulated in Matlab and implemented in FPGA devices. The experimental results obtained show that iATA is more memory efficient than the best existing tables-and-additions methods. For the logarithm function, 87.22% memory size reduction is observed when it is compared with the best available result by SBTM. For the reciprocal function, it achieves a memory saving of 23.63% when compared to the best existing result obtained by MTM. For the natural logarithm function, iATA reduces the memory size by 61.39% when compared to the best existing result obtained by STAM.

CHAPTER 6. Discussions, Conclusions and Future Works

6.1 Discussions

For better illustration of the trade-off between accuracy and hardware merits, the *ADP* (computed from synthesis results as tabulated in Table 3-10, Table 4-7 and Table 5-4) of the designs (logarithmic converters) obtained by the proposed UMA, RM and iATA are plotted together in Fig. 6-1. From Fig. 6-1, two key observations can be resulted. The foremost, the Mitchell-based methods are far more superior in terms of circuit size and delay for the logarithmic conversion. Secondly, the *ADP* of the designs increases as the accuracy requirement increases, indicating that the design becomes more complex with higher accuracy requirement. This suggests that a LNS-based system should consider logarithmic converters with accuracy matching the requirements of the application systems. It is worth to highlight that the Mitchell-based UMA and RM methods are able to capitalize on the relaxation of accuracy requirements to achieve significantly optimized designs. However, they are only scalable up to 18 bits accuracy in the current implementation. For accuracy requirements greater than 18 bits, the iATA method of the tables-and-addition method is inevitably needed.

6.2 Conclusions

The existing methods for logarithmic conversion over a range of accuracies have been optimized for efficient hardware implementation as summarized below.

The advantageous features of Mitchell-based methods are enhanced and extended to higher accuracy by a novel method named the Unified Mitchell-based Approximation (UMA) which is proposed in this research to obtain an optimized circuit for any desired logarithmic conversion accuracy. The proposed method explores a design space consisting of five design parameters that impact the accuracy and hardware merits. As the design space of the five parameters is too huge for exhaustive search, constraints are proposed for the parameters and a new search algorithm is proposed to sweep across the range of the parameter values so that the search can be completed within practical period of time without compromising the optimization objective. The designs obtained by UMA are the same as or better than the corresponding thirteen existing Mitchell-based designs. Eight of the thirteen designs obtained by the proposed UMA achieve improvement by more than 50% in area-delay product. Apart from being optimized, UMA is scalable and able

to find an optimized circuit for any accuracy requirement up to 14 bits while the existing Mitchell-based methods achieve accuracies up to less than 12 bits.

Fig. 6-1: Area-delay product for logarithmic converters obtained by the proposed methods for accuracy requirements ranging from 4 bits to 24 bits

To capitalize the advantageous features of the Mitchell-based methods, a novel approach named Range Mapping (RM) is proposed for accuracies beyond 14 bits, which are not able to be achieved by UMA. The proposed RM method compresses the range of approximation to less than

TABLE 6-1: SUMMARY ON TARGETED AIMS AND RESULTANT ACHIEVEMENTS IN THIS WORK ON LOGARITHMIC CONVERSION

Objectives	Performance Metrics	Minimum Targets	Achievements
1	Area-delay product (Mitchell-based)	Reduced by 50% for at least 50% of the existing designs	Reduced by more than 50% for 8 of the 13 existing designs
2	Highest accuracy (Mitchell-based)	Improved by 4 bits from 12 to 16 bits	Improved by 6 bits from 12 to 18 bits
3	Table size (Tables-and-additions based)	Reduced by 50%	Reduced by 87.22%

one third of the Mitchell range. A search algorithm is developed to determine the compressed range such that high accuracy approximation can be achieved with minimum hardware and delay overhead. After range mapping, the compressed range is smaller, more linear and has fewer Mitchell fractions. With these properties, when a newly proposed 4-region piecewise linear approximation is developed heuristically for the compressed range, the proposed method is able to reduce the maximum absolute error and maximum absolute error percentage of the binary logarithm computation by 15.0% and 25.0% respectively when compared to the best existing results obtained by Kim et al. [Kim06] and De Caro et al. [DeC11] respectively. The proposed figures of merit indicate that the *ADP* overhead introduced by the proposed method in achieving high accuracy is justifiable. Therefore, when the proposed UMA is applied on the compressed range, the three advantageous properties facilitate the search for an optimized piecewise linear approximation function and, hence, higher accuracy (15 bits – 18 bits) logarithmic converters can be obtained.

A new tables-and-additions method, named as iATA, for high accuracy computation of elementary functions is proposed. The proposed method formulates the approximation equation without using the central difference formulation. It integrates three techniques, namely the carry select technique, the symmetry property exploitation and the unequal partitioning of the input with the aid of error analysis. iATA is more memory efficient than the best existing tables-and-additions methods. For the logarithm function, 87.22% memory size reduction is observed when it is compared with the best available design by SBTM. For the reciprocal function, it achieves a memory saving of 23.63% when compared to the best existing result obtained by MTM. For the natural logarithm function, iATA reduces the memory size by 61.39% when compared to the best existing result obtained by STAM. Being an elementary function computation method, the proposed iATA can be employed to compute binary logarithm of any accuracy up to 24 bits.

Table 6-1 summarizes the targeted aims and achievements in this work. For the work on the Mitchell-based methods, 8 of the 13 designs obtained by the proposed UMA achieve improvement in *ADP* by more than 50% as compared to the corresponding existing designs while the target set

was 50% or more for 50% of the known existing designs. In term of accuracy of the Mitchell-based methods, this work aims at an improvement of at least 4 bits and has successfully achieved an improvement of 6 bits through the combination of the proposed RM and UMA. Lastly, reducing the table size of the tables-and-additions methods by at least 50% is another aim of this work and a reduction of 87.22% for the computation of logarithm by the proposed iATA is achieved.

From this work and the experimental results obtained, it is evident that the Mitchell-based methods are highly efficient for the implementation of logarithmic conversion in integrated circuits. The UMA method and the RM method developed in this work are able to achieve the architectures with the *ADP* optimized and they have extended the conversion accuracy from the current less than 12 bits to 18 bits. The iATA method developed improves the existing efficient LUT-based methods for high conversion accuracy of 20 bits and above. It is believed that the Mitchell-based methods can be further developed and improved for higher accuracy and smaller *ADP*. The LUT-based methods can also be investigated for further reduction in table size. Some potential future works are suggested in the next section.

6.3 Future Works

Based on the research presented in this thesis, several potential research directions and ideas have been identified.

(1) The systematical search algorithm introduced in the proposed UMA in Chapter 3 is able to obtain an optimized approximation function (design) within practical period of time for accuracy requirements up to 14 bits and 18 bits, respectively, with and without the proposed RM method. Two main issues that lead to long search time, and hence impeding the search for higher accuracy design, are identified as follows. Firstly, in the search for high accuracy designs, the number of bits used in the error analysis (Section 3.7) for the input N needs to be increased. As a result, the search time will increase significantly as the error analysis is performed for each possible gradient values, C_r in (3.13), for each f-regions until all the f-regions become s-regions. One of the potential solutions is to develop an error estimation method that provides an upper bound of the error for the approximation function instead of computing the exact error for each N value. The second issue is due to the impractically large number of combinations of valid gradient values from each region that need to be evaluated (Section 3.8) for every value of the design parameter P . To overcome this problem, heuristic and/or greedy approach can be employed to filter off as many as possible the valid gradient values of each region before the evaluation of hardware merits.

(2) In Chapter 4, the proposed range mapping (RM) method compresses the Mitchell range $M = [1, 2)$ into a smaller range, named the focus range $F = [1.25, 1.5625)$, which possesses three advantageous properties that facilitate a higher accuracy approximation of binary logarithmic function by the piecewise linear approximation. For even higher accuracy approximation, a new focus range can be sought for even better properties in term of the size of the range, its location and possibly other new properties. To achieve this objective, several research areas can be investigated. In this work, the mapping constants G_r are constrained to each contain at most two power-of-two terms (POTTs) to reduce the complexity of the circuit implementing the mapping. By relaxing the number of the POTTs to more than two, the flexibility of the constants G_r can be increased and hence a better focus range could be found. Further research can also be carried out to develop a better algorithm in place of **find_F(S, I, Q)** in Fig. 4-2.

(3) In the proposed iATA method described in Chapter 5, two sets of design parameters S_A and S_B have been proposed in Section 5.3.2. Compared to S_A , S_B leads to a more efficient hardware implementation. In other words, there may be other feasible sets of design parameters which can achieve better hardware performances as compared to S_B without compromising the accuracy requirement. In this work, the two parameters in S_1 and S_2 are obtained heuristically based on the error analysis. To find better sets of design parameters, the mixed integer linear programming (MILP) and/or other systemic searching approaches can be applied. The possible constraints for the MILP are the error expressions and conditions described in Section 5.3.2 as well as table size equations.

(4) The work in this thesis shows that the relaxation of logarithmic conversion accuracy can be translated to simplified logarithmic converter. Not only can this concept of trading off accuracy and hardware complexity be applied to other arithmetic units but it can also be applied to large systems or applications, for example, digital filter designs.

(5) Last, but not least, the Mitchell-based methods could be further developed for the computation of other arithmetic functions. Some preliminary works [Low12b] [Low13d] on the computation of the square root and division show that the Mitchell-based methods maintain its efficiency in hardware implementation but still suffer from good accuracy. It is believed that much more works can be done on the computation of arithmetic functions using the Mitchell-based methods.

Author's Publications

Journal Publications

- [Low13b] **J. Y. L. Low**, and C. C. Jong, "A memory efficient tables-and-additions method for accurate computation of elementary functions," *IEEE Transactions on Computers*, vol. 62, no.5, pp. 858-872, 2013.

Conference Publications

- [Low13d] **J. Y. L. Low**, C. C. Jong, J. Y. S. Low, "Non-iterative high speed division computation based on Mitchell logarithmic method" ,” in *Proc. 2013 IEEE Int. Symp. on Circuits and Systems (ISCAS-2013)*, 2013.
- [Low12b] **J. Y. L. Low**, C. C. Jong, J. Y. S. Low, T. F. Tay and C. H. Chang, "A fast and compact circuit for integer square root computation based on Mitchell logarithmic method,” in *Proc. 2012 IEEE Int. Symp. on Circuits and Systems (ISCAS-2012)*, pp. 1235-1238, 2012.
- [Low10] **J. Y. L. Low**, and C. C. Jong, "High accuracy binary logarithmic conversion using range mapping for DSP applications,” in *Proc. 2010 IEEE Asia Pacific Conf. on Circuits and Systems (APCCAS-2010)*, pp. 332-335, 2010.

Bibliography

- [Abe03] K.H. Abed and R.E. Siferd, "CMOS VLSI implementation of a low-power logarithmic converter," *IEEE Trans. Computers*, vol. 52, no. 11, pp. 1421-1433, 2003.
- [Ali10] A. Alimohammad, S.F. Fard and B.F. Cockburn, "A Unified Architecture for the Accurate and High-throughput Implementation of Six Key Elementary Functions," *IEEE Trans. Computers*, vol. 59, no. 4, pp. 449 - 456, 2010.
- [Arn90] M.G. Arnold, T.A. Bailey, J.R. Cowles and J.J. Cupal, "Redundant logarithmic arithmetic," *IEEE Trans. Computers*, vol. 39, no. 8, pp. 1077-1086, 1990.
- [Arn07] M.G. Arnold and P.D. Vouzis, "A serial logarithmic number system ALU," *10th Euromicro Conf. Digital System Design Architectures, Methods and Tools (DSD '07)*, pp. 151-156, 2007.
- [Ash10] A. Ashrafi, R. Adhami and A. Milenkovic, "A Direct Digital Frequency Synthesizer Based on the Quasi-linear Interpolation Method," *IEEE Trans. Circuits and Systems-I: Regular Papers*, vol.57, pp. 863-872, 2010.
- [Cao01] J. Cao, B.W.Y. Wei and J. Cheng, "High-performance Architectures for Elementary Function Generation," *Proc. 15th IEEE Symp. Computer Arithmetic (ARITH-15 '01)*, pp. 136-144, 2001.
- [Com65] M. Combet, H. Van Zonneveld and L. Verbeek, "Computation of the base two logarithm of binary numbers," *IEEE Trans. Computers*, vol. EC-14, pp. 863-867, 1965.
- [Dad65] L. Dadda, "Some changes for parallel multipliers," *Alta Frequenza*, vol. 34, pp. 349-356, 1965.
- [Das95] D. Das Sarma and D.W. Matula, "Faithful Bipartite ROM Reciprocal Tables," *Proc. 12th Symp. Computer Arithmetic (ARITH-12 '95)*, pp. 17-28, 1995.
- [DeC09] D. De Caro, N. Petra and A. Strollo, "Digital Synthesizer/Mixer with Hybrid CORDIC - Multiplier Architecture: Error Analysis and Optimization," *IEEE Trans. Circuits and Systems-I: Regular Papers*, vol. 56, pp. 364-373, 2009.

- [DeC11] D. De Caro, N. Petra and A.G.M. Strollo, "Efficient Logarithmic Converters for Digital Signal Processing Applications", *IEEE Trans. Circuits and Systems-II: Express Briefs*, vol. 58, no. 10, pp. 667-671, 2011.
- [deD05] F. de Dinechin and A. Tisserand, "Multipartite Table Methods," *IEEE Trans. Computers*, vol. 54, no. 3, pp. 319-330, 2005.
- [Det05] J. Detrey and F. de Dinechin, "Table-based Polynomials for Fast Hardware Function Evaluation," *Proc. IEEE International Conf. Application-Specific Systems, Architecture Processors (ASAP '05)*, pp. 328-333, 2005.
- [Gup13] V. Gupta, D. Mohapatra, A. Raghunathan and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124-137, 2013.
- [Gut11] R. Gutierrez and J. Valls, "Low Cost Hardware Implementation of Logarithm Approximation," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 12, pp. 2326 - 2330, 2011.
- [Hal70] E.L. Hall, D.D. Lynch and S.J. Dwyer, "Generation of products and quotients using approximate binary logarithms for digital filtering applications," *IEEE Trans. Computers*, vol. C-19, no. 2, pp. 97-105, 1970.
- [Hsi08] S.-F. Hsiao, P.-C. Wei and C.-P. Lin, "An Automatic Hardware Generator for Special Arithmetic Functions Using Various ROM-based Approximation Approaches," *Proc IEEE International Symp. Circuits and Systems (ISCAS '08)*, pp. 468-471, 2008.
- [Hua08] P. Huang, D.H.-Y Teng, K. Wahid and S.-B Ko, "Convergence analysis of Jacobi iterative method using logarithmic number system," *7th IEEE/ACIS International Conf. Computer and Information Science (ICIS '08)*, pp. 27-32, 2008.
- [Jua09] T.-B. Juang, S.-H. Chen and H.-J. Cheng, "A lower error and ROM-free logarithmic converter for digital signal processing applications," *IEEE Trans. Circuits and Systems-II: Express Briefs*, vol. 56, no. 12, pp. 931-935, 2009.
- [Kar01a] K. Karagianni, V. Paliouras, G. Diamantakos and T. Stouraitis, "Operation-Saving VLSI Architectures for 3D Geometrical Transformations," *IEEE Trans. Computers*, vol. 50, pp. 609-622, 2001.

- [Kar01b] K. Karagianni and T. Stouraitis, "A Vector Processor for 3-D Geometrical Transformations," *Proc. IEEE International Symp. Circuits and Systems, (ISCAS '01)*, vol. 4, pp. 482-485, 2001.
- [Kim06] H.-J. Kim, B.-G. Nam, J.-H. Sohn, J.-H. Woo and H.-J. Yoo, "A 231-MHz, 2.18-mW 32-bit logarithmic arithmetic unit for fixed-point 3-D graphics system," *IEEE J. Solid-State Circuits*, vol. 41, no. 11, pp. 2373-2381, 2006.
- [Lai02] M.-H. Lai, M.-F. Yu and S.-G. Chen, "An Efficient Modified Phong Shading Algorithm & its Low-complexity Realization," *Proc. IEEE International Symp. Circuits and Systems (ISCAS '02)*, vol. 4, pp. IV-201 – IV-204, 2002.
- [Lam03] P. Lamarche and Y. Savaria, "VHDL Source Code Generator and Analysis Tool to Design Linear Interpolators," *Proc. First IEEE Northeast Workshop Circuits and Systems*, pp. 67-72, 2003.
- [Lan05] T. Lang and E. Antelo, "High-throughput CORDIC-based Geometry Operations for 3D Computer Graphics," *IEEE Trans. Computers*, vol. 54, pp. 347-361, 2005.
- [Lee05a] P. Lee, "An evaluation of a hybrid-logarithmic number system DCT/IDCT algorithm" *IEEE International Symp. Circuits and Systems (ISCAS '05)*, vol. 5, pp. 4863-4866, 2005.
- [Lee05b] D.-U. Lee, A.A. Gaffar, O. Mencer and W. Luk, "Optimizing Hardware Function Evaluation," *IEEE Trans. Computers*, vol. 54, pp. 1520-1531, 2005.
- [Lee07] D.-U. Lee and J.D. Villasenor, "A bit-width Optimization Methodology for Polynomial-based Function Evaluation," *IEEE Trans. Computers*, vol. 56, pp. 567 - 571, 2007.
- [Lee08] D.-U. Lee, R.C.C. Cheung, W. Luk and J.D. Villasenor, "Hardware Implementation Trade-offs of Polynomial Approximations and Interpolations," *IEEE Trans. Computers*, vol. 57, pp. 16, 2008.
- [Lew94] D. Lewis, "Interleaved Memory Function Interpolators with Application to an Accurate LNS Arithmetic Unit," *IEEE Trans. Computers*, vol. 43, pp. 974-982, 1994.

- [Li08] Z. Li, J. An, M. Yang and J. Yang, "FPGA design and implementation of an improved 32-bit binary logarithm converter," *4th International Conf. Wireless Communications, Networking and Mobile Computing (WiCOM '08)*, pp. 1-4, 2008.
- [Liu00] P. Liu and S.N. Bhatt, "Experiences with Parallel N-body Simulation," *IEEE Trans. Parallel and Distributed Systems*, vol. 11, pp. 1306-1323, 2000.
- [Lo87] H.-Y. Lo and J.-L. Chen, "A hardwired generalized algorithm for generating the logarithm for base-k by iteration," *IEEE Trans. Computers*, vol. C-36, no. 11, pp. 1363-1367, 1987.
- [Low12a] J. Y. S. Low and C. H. Chang, "A VLSI efficient programmable power-of-two scaler for $\{2^{n-1}, 2^n, 2^{n+1}\}$ RNS," *IEEE Trans. Circuits and Systems-I: Regular*, 2012, vol. 59, no. 12, pp. 2911-2919, 2012.
- [Maj73] J.C. Majithia and D. Levan, "A note on base-2 logarithm computations," *Proc. the IEEE*, vol. 61, no. 10, pp. 1519-1520, 1973.
- [McC03] J.M. McCollum, J.M. Lancaster, D.W. Bouldin and G.D. Peterson, "Hardware Acceleration of Pseudo-random Number Generation for Simulation Applications," *Proc. 35th Southeastern Symp. System Theory (SSST-35 '03)*, pp. 299-303, 2003.
- [Meh09] P.K. Meher, J. Valls, T.-B. Juang, K. Sridharan and K. Maharatna, "50 years of CORDIC: algorithms, architectures and applications," *IEEE Trans. Circuits and Systems-I: Regular*, vol. 56, no. 9, pp. 1893-1907, 2009.
- [Mit62] J.N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Trans. Electronic Computers*, vol. EC-11, no. 4, pp. 512-517, 1962.
- [Mul98] J.-M. Muller, A. Scherbyna and A. Tisserand, "Semi-logarithmic number systems," *IEEE Trans. Computers*, vol. 47, no. 2, pp. 145-151, 1998.
- [Mul99] J.M. Muller, "A Few Results on Table-based Methods," *Reliable Computing*, vol. 5, pp. 279-288, 1999.
- [Nam08] B.-G Nam, H.-J. Kim and H.-J. Yoo, "Power and area-efficient unified computation of vector and elementary functions for handheld 3D graphics systems," *IEEE Trans. Computers*, vol. 57, no. 4, pp. 490-504, 2008.

- [Nam09] B.-G. Nam and H.-J. Yoo, "An embedded stream processor core based on logarithmic arithmetic for a low-power 3-D graphics SoC," *IEEE J. Solid-State Circuits*, vol. 44, no. 5, pp. 1554-1570, 2009.
- [NVI11] NVIDIA, "Tesla C2075 Companion Processor," Tesla C2075 datasheet, 2011.
- [Pal00] V. Paliouras, K. Karagianni and T. Stouraitis, "A Floating-point Processor for Fast and Accurate Sine/Cosine Evaluation," *IEEE Trans. Circuits and Systems-II: Analog and Digital Signal Processing*, vol. 47, pp. 441-451, 2000.
- [Phi96] G.M. Philips and P.J. Taylor, *Theory and Applications of Numerical Analysis*, Academic Press, 1996.
- [Pin04] J.-A. Pineiro, M.D. Ercegovic and J.D. Bruguera, "Algorithm and Architecture for Logarithm, Exponential, and Powering Computation," *IEEE Trans. Computers*, vol. 53, pp. 1085-1096, 2004.
- [Pin05] J.-A. Pineiro, S.F. Oberman, J.-M. Muller and J.D. Burguera, "High-speed Function Approximation Using a Minimax Quadratic Interpolator," *IEEE Trans. Computers*, vol. 54, pp. 304-318, 2005.
- [Rod10] T.K. Rodrigues and E.E. Swartzlander, "Adaptive CORDIC: Using Parallel Angle Recoding to Accelerate Rotations," *IEEE Trans. Computers*, vol. 59, no. 4, pp. 522-531, 2010.
- [San99] S.L. SanGregory, C. Brothers, D. Gallagher and R. Siferd, "A fast, low-power logarithm approximation with CMOS VLSI implementation," *42nd Midwest Symp. Circuits and Systems, (MWSCAS '99)*, vol. 1, pp. 388-391, 1999.
- [Sas07] T. Sasao, S. Nagayama and J.T. Butler, "Numerical Function Generators Using LUT Cascades," *IEEE Trans. Computers*, vol. 56, pp. 826-838, 2007.
- [Sch99] M.J. Schulte and J.E. Stine, "Approximating Elementary Functions with Symmetric Bipartite Tables," *IEEE Trans. Computers*, vol. 48, no. 8, pp. 842-847, 1999.
- [Shi01] H.-C. Shin, J.-A. Lee and L.-S. Kim, "A Hardware Cost Minimized Fast Phong Shader," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 9, pp. 297-304, 2001.

- [Sti99] J.E. Stine and M.J. Schulte, "The symmetric table addition method for accurate function approximation," *J. VLSI Signal Processing*, vol. 21, no. 2, pp. 167-177, 1999.
- [Sto01] T. Stouraitis and V. Palioras, "Considering the alternatives in low-power design," *IEEE Circuitis and Devices Magazine*, vol. 17, no. 4, pp. 22-29, 2001.
- [Str07] A.G.M. Strollo, D. De Caro and N. Petra, "A 630 MHz, 76 mW Direct Digital Frequency Synthesizer Using Enhanced ROM Compression Technique," *IEEE J. Solid State Circuits*, vol. 42, pp. 350-360, 2007.
- [Str08] A. Strollo, D. De Caro and N. Petra, "A 430 MHz, 280 mW Processor for the Conversion of Cartesian to Polar Coordinates in 0.25 μ m CMOS," *IEEE J. Solid State Circuits*, vol. 43, pp. 2503-2513, 2008.
- [Str11] A.G.M. Strollo, D. De Caro and N. Petra, "Elementary functions hardware implementation using constrained piecewise-polynomial approximations," *IEEE Trans. Computers*, vol. 60, no. 3, pp. 418 - 432, 2011.
- [Sum08] M.G.B. Sumanasena, "A Scale Factor Correction Scheme for the CORDIC Algorithm," *IEEE Trans. Computers*, vol. 57, pp. 1148-1152, 2008.
- [Swa75] E.E. Swartzlander Jr. and A.G. Alexopoulos, "The sign/logarithm number system," *IEEE Trans. Computers*, vol. C-14, no. 12, pp. 1238-1242, 1975.
- [Swa83] E.E. Swartzlander Jr., D.V. Satish Chandra, H.T. Nagle Jr., and S.A. Starks, "Sign/logarithm arithmetic for FFT implementation," *IEEE Trans. Computers*, vol. C-32, no. 6, pp. 526-534, 1983.
- [Tak98] N. Takagi, "Powering by a Table Look-up and a Multiplication with Operand Modification," *IEEE Trans. Computers*, vol. 47, pp. 1216-1222, 1998.
- [Ver02] H. T. Vergos, C. Efstathiou and D. Nikolos, "Diminished-one modulo 2^n+1 adder design," *IEEE Trans. Computers*, vol. 51, pp. 1389-1399, 2002.
- [Vol59] J.E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Trans. Electronic Computers*, vol. EC-8, no. 3, pp. 330-334, 1959.

- [Wal05] E.G. Walters III and M.J. Schulte, "Efficient Function Approximation Using Truncated Multipliers and Squarers," *Proc. 17th IEEE Symp. Computer Arithmetic (ARITH-17 '05)*, pp. 232-239, 2005.
- [Won95] W.F. Wong and E. Goto, "Fast Evaluation of the Elementary Functions in Single Precision," *IEEE Trans. Computers*, vol. 44, pp. 453-457, 1995.
- [Xil05] Xilinx, "FPGA Design Tips," http://www.xilinx.com/itp/xilinx7/books/data/docs/dev/dev0017_5.html, 2005.
- [Yos98] K. Yoshida, T. Sakamoto and T. Hase, "A 3D graphics library for 32-bit microprocessors for embedded systems," *IEEE Trans. Consumer Electronics*, vol. 44, no. 3, pp. 1107-1114, 1998.