2012

Improved MDLNS Number System Addition and Subtraction by Use of the Novel Co-Transformation

LEILA SEPAHI
University of Windsor

Improved MDLNS Number System Addition and Subtraction by Use of the Novel Co-Transformation

by

Leila Sepahi

A Thesis
Submitted to the Faculty of Graduate Studies
through Electrical and Computer Engineering
in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada

2012

Improved MDLNS Number System Addition and Subtraction by Use of the Novel Co-Transformation

by

Leila Sepahi

APPROVED BY:

Dr. Edwin Tam
Department of Civil and Environmental Engineering

Dr. Huapeng Wu
Department of Electrical and Computer Engineering

Dr. Roberto Muscedere, Advisor
Department of Electrical and Computer Engineering

Dr. Mitra Mirhassani, Chair of Defense
{Select department/faculty}

May 10, 2012

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Multi-Dimensional Logarithmic Number System (MDLNS) is a generalized version of the Logarithmic Number System (LNS) which has multiple dimensions or bases. These generalizations can increase accuracy and hardware efficiency. However, addition and subtraction operations are the major obstruction of all logarithmic number systems circuits and so far a fair amount of research has been done to find practical techniques in LNS to implement these operations efficiently without the need for large tables. In order to achieve this goal, several methods such as interpolation, multipartite tables, and co-transformation have been introduced to decrease the cost and complexity. One of the most recent works is Novel Co-transformation.

This thesis investigates the application of the Novel Co-Transformation on MDLNS. The goal is to reduce the table sizes over previously published method which utilizes a different address decoder on its tables which requires greater overhead. The results show that the table sizes are reduced significantly when a minimal error is allowed. Other common LNS techniques for table reductions may be applied to obtain better results.

DEDICATION

To My devoted Parents

and

To My Best Sister for her support.

ACKNOWLEDGEMENTS

Through this acknowledgment, I would like to express my sincere gratitude to my supervisor Dr.Muscedere for his constant support and invaluable guidance. He gave me this opportunity to learn the subject throughout the course of this thesis work.

I express my thanks to the committee members Dr.Tam and Dr.Wu who have shared their opinions and valuable suggestions regarding the project to make it a worthwhile experience.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DSP	Digital Signal Processor
LNS	Logarithmic Number System
LUT	Look Up Table
MDLNS	Multi-Dimensional Logarithmic Number System
RALUT	Range Addressable Look Up Table
SBD	Single Base Domain

CHAPTER I

INTRODUCTION

Introduction

All microelectronic devices consist of integrated circuits which contain a huge number of interconnected transistors. Microprocessors, for example, are an integrated circuit that can perform all the logic and mathematical functions and works as the central processing unit of a generalized computer. Although modern microprocessors can process significant amounts of information in a short amount of time, they are not the best choice for “embedded” systems such as mobile or ubiquitous devices. .Digital Signal Processors are a most practical choice as they are specifically designed to perform the necessary tasks of managing digital signal processing (DSP) using very streamlined mathematical calculations while meeting specifications and remaining in a very small foot print which is ideal for mobile devices [2]. DSP is the basis for all modern digital communication.

DSP itself has been a driver for many applications of alternative number representations through which a considerable amount of research has been performed to optimize performance during the last couple of decades [2]. In most DSP applications, multipliers are one of the most resource (space, speed and latency) consuming fundamental units. Hence a more optimal multiplier results in a more efficient device. In any hardware design there are always technical trade-offs among area, latency and accuracy [2] [3].

Number Systems

Numbers in these computation processors can be stored and processed in a variety of formats, the most common being fixed-point and floating-point number systems [2]. Fixed point is the application of a basic binary representation with the assumption that a "decimal point" appears at a fixed place in all the numbers. For example, the integer 488362 can be interpreted as a fixed point number if it is assumed the decimal is 10 bits, so that $488362/2^{10} = 476.916015625$. Although the fractional part of this number appears accurate, the next higher possible fixed point value for representation is $488363/2^{10} = 476.9169921875$, a difference of $1/2^{10} = 0.0009765625$. This increment may not be small enough for a given application or does not provide enough resolution. In order to increase it, one only needs to increase the number of bits for the decimal or fractional portion, but this may come at a cost of more hardware (in custom systems) or require a new architecture in ready-to-use solutions (moving from 16-bit to 32-bit or to 64-bit processor class). This lack of a high dynamic range makes fixed-point number systems adequate for a subset of applications as the hardware is less costly and the accuracy requirement may be acceptably low [2].

The floating-point number system (FPNS), an extension of the fixed point number system, uses two integers respectively, the mantissa and exponent to form the individual word. The exponent allows for an increase in the dynamic range while still retaining the numerical accuracy provided by the mantissa portion. This offers better precision than the fixed-point number system but at an additional hardware cost in terms of both area and delay. Seemingly simple operations such as addition and subtraction require de-normalization and normalization steps (shifting) to ensure the representation stays correct. [5][4][2].

When dealing with any integer binary representation, multiplication operations are slower (longer latencies) and larger and therefore treated as penalties compared to addition and subtraction. This penalty is the basis for exploring alternative number system which can reduce the impact of multiplication on a circuit.

The Logarithmic Number System (LNS) is an alternative variation of floating point for representing real numbers in digital hardware especially for DSP applications. A number is represented in the form of 2^x, where x is in a fixed-point reorientation. The main benefit of LNS is that it simplifies the hardware required for the operations of multiplication, division, powers and roots to same scale of addition, subtraction, multiplication and division, respectively for binary systems [6][1]. Unfortunately, simple operations in LNS such as addition and subtraction are much more difficult to implement as they require the use of large non-linear tables.

Numerous studies have compared floating-point number system against LNS in particular applications. LNS can outperform floating-point in terms of smaller word sizes versus error performance.

A more generalized version of LNS is Multi-Dimensional Logarithmic Number System (MDLNS) which offers the ability to use multiple digits and orthogonal bases to improve representation space while reducing table complexity. It still however has some of LNS's problems such as addition and subtraction.

Since LNS has shown significant promise in a field of applications, during the past few decades it has been tried to alleviate these problems. Particularly for additional and subtraction, a variety of table methods have been introduced such as interpolation, multipartite tables, and co-transformation which have incrementally reduced the traditionally large footprint to more manageable sizes. This work aims specifically to apply one of these latest techniques (Novel Co-Transformation) to MDLNS to further reduce addition and subtraction circuit implementations.

Thesis Organization

The organization of our work in this thesis is as follows: Chapter 2 will briefly review different existing number systems. Background knowledge on certain related number systems is provided and both the benefits and shortcomings of each system are discussed. After this brief review, Chapter 3 will focus on the newest number systems, LNS and MDLNS and the problem of Addition and Subtraction in LNS. Then our proposed method of improvement for MDLNS will be discussed. Chapter 4 is the results of the work which will be consisted of comparative results from the designed MATLAB code and the results of previous methods. And finally Chapter 5 will go through the conclusion of the work and some suggestions for future work. Also all of the designed MATLAB codes can be found in Appendix A.

CHAPTER II

REVIEW OF THE RELATED NUMBER SYSTEMS AND MDLNS IMPLEMENTATIONS

Introduction

This chapter will review, in brief, the most common number systems used in computing that are relevant to this thesis as well as the most significantly relevant methods of addition and subtraction in the LNS and MDLNS domains. References are included to provide more information if the reader requires.

Floating Point Number System (FPNS)

Unlike fixed-point number representation, FPNS has larger dynamic range (exponent b), and better precision (mantissa a). The first digit is always assumed to be a one, unless when x = 0 which is a special case.

$$x = 1.a \times 2^{b}$$

Both of these qualities are defined by an integer with a certain number of bits available to represent each. If a higher range is required, more bits can be used to represent the exponent portion where as if higher precision is requires, more bits can be used to represent the mantissa. In either case, adding more bits results in a larger and slower circuit. In general, FPNS is defined by a standard number of bits to allow for interoperability between different processor and platform types. For example, Intel and PowerPC processors are quite different, but the encoding of FPNS data is identical. For some applications, a FPNS may offer too much precision and dynamic range and therefore the resulting hardware would be excessive for the needs of the system. One may consider a fixed-point system instead. Although floating point offers good precision, its implementation requires more steps, such as de-normalization, normalization and rounding, as the decimal point needs to be compensated for all operations. In some cases, a 32-bit fixed-point system may be chosen over a 32-bit FPNS as it is simpler to use and implement.

Logarithmic Number System

A typical DSP system is based on the multiplication and accumulation (addition) of many coefficients with some real world input data. These systems generally do not favor or disfavor particular operations. When an implementation is chosen, a designer may take an optimization approach that will favor a particular operation in order to reduce a particular resource. Depending on the ratio of multiplication over addition and subtraction operations in a system, one can use an LNS representation. LNS, in some applications, is more efficient in terms of area which requiring a fewer number of bits and consequently results in a decreased latency of the circuit compared to a binary system, while achieving the same error performance [3][7].

In LNS, the representation is controlled completely by the exponents. As with FPNS, $x = 0$ is a special case.

$$x = (-1)^s \times b^a \tag{2.1}$$

In Eq. 2.1, s is the sign of X (s = 0 if $X > 0$ and $s = 1$ if $X < 0$) and a is a generally a binary two's complement fixed-point representation with k integer bits and f fractional bits. The simplicity of the representation demonstrates the advantages especially with multiplication, division, and exponents as they are reduced to addition, subtraction and multiplication on the exponents (smaller word size) respectively. Unfortunately, the simple operations in binary arithmetic are the most difficult in LNS such as addition and subtraction; which may require the use of larger non-linear calculations depending on the sizes of k and *f*. To this, a considerable amount of research has been conducted over the years to mitigate the LNS addition and subtraction problem and overall improve the number system.

Multi-Dimensional Logarithmic Number System (MDLNS)

The Multi-Dimensional Logarithmic Number System (MDLNS) is a generalized version of the LNS. It utilizes multiple orthogonal bases as well as the ability to use multiple digits which can introduce redundancy into the system and reduce the hardware complexity compared to LNS. Unfortunately, there is no monotonic relationship between standard linear representations and MDLNS representations as there is in LNS. This makes the process of conversion from binary as well as addition and subtraction slightly more difficult [1].

$$x = \sum_{i=1}^{n} s_i \cdot \prod_{j=1}^{k} D_j^{\,b_{i,j}} \tag{2.2}$$

In Eq.2.2 k is the number of bases used (at least two), s_i is sign of each digit {–1, 0, +1}, D_j is base and can be a real number. The first base, D_1, will always be assumed to be 2, $b_{i,j}$ are integer powers for base j of digit i.

The use of multiple bases allows for smaller ranges on the non-binary exponents ($D_{2\to k}$) which can yield to the same precision as LNS but with fewer bits. It is also possible to select the bases such that a particular set of numbers can be represented with minimal quantization error [8]. This approach allows the system to be smaller while still retaining a higher level of accuracy compared to similar sized LNS. All of these advantages make MDLNS a possible alternative number system for some applications [1].

Classic Method of Addition/Subtraction in LNS

To perform the addition and subtraction in LNS, the classic method is to use multiplication of one of the addends with a factor. Depending on the sign of *z*, we will multiply either the largest or smallest of the addends (*X*) by a factor S_b (or D_b for subtraction). These factors are derived below and are shown graphically in figures 2.1 and 2.2.

$$\mathrm{X} + \mathrm{Y} = \mathrm{X}\left(1 + \frac{Y}{X}\right) = X(1 + Z) \overset{LNS}{\longrightarrow} x + S_b(z)$$

$$S_b(z) = \log_b(1 + b^z)$$

The constant b is the base of the logarithms, mostly assumed to be 2 to simplify circuit implementation.

$$\mathrm{X} - \mathrm{Y} = \mathrm{X}\left(1 - \frac{Y}{X}\right) = X(1 - Z) \overset{LNS}{\longrightarrow} x + D_b(z)$$

$$D_b(z) = \log_b(1 - b^z)$$

For Addition/Subtraction with $z > 0$:

$$\log_b(|X| + |Y|) = \min(x, y) + S_b(|x - y|)$$

$$\log_b(||X| - |Y||) = \min(x, y) + D_b(|x - y|) \qquad (2.3)$$

For Addition/Subtraction with $z < 0$:

$$\log_b(|X| + |Y|) = \max(x, y) + S_b(-|x - y|) \qquad (2.4)$$

$$\log_b(||X| - |Y||) = \max(x, y) + D_b(-|x - y|)$$

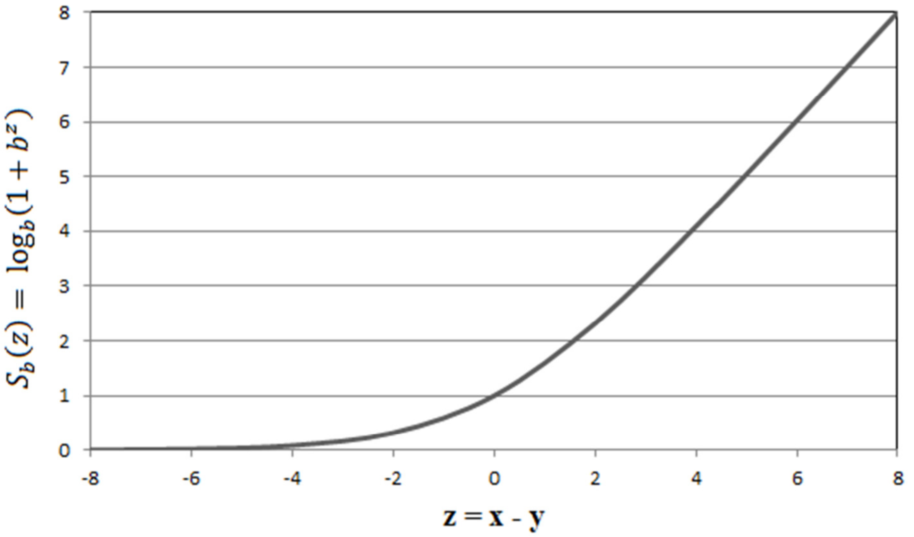

Figure 2.1: LNS Addition Relationship for D =2

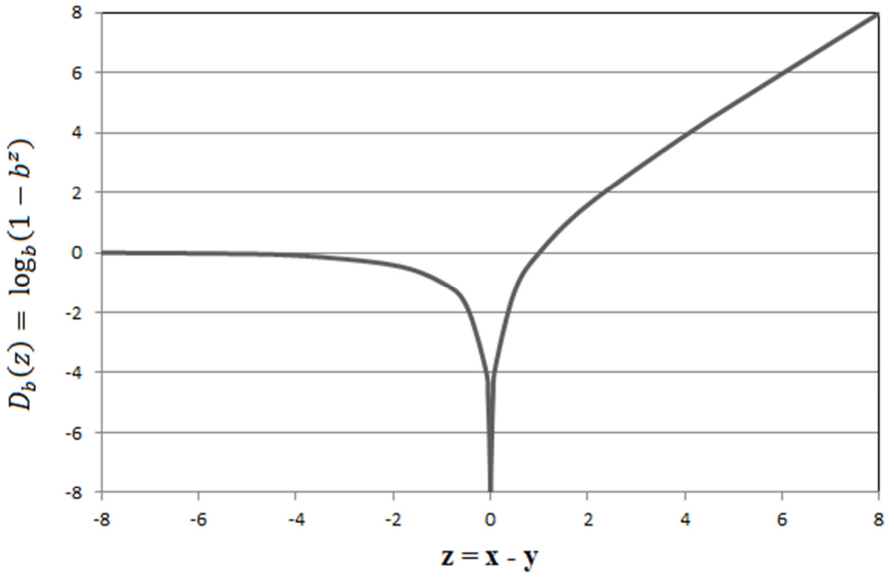

Figure 2.2: LNS Subtraction Relationship for D=2

LNS Implementation

Up to now in literature, several different number representations have been introduced to implement LNS addition and subtraction in hardware [9] [10] [11] [12] [13] (integer, fixed-point, floating-point and integer rational numbers). Depending on the method of implementation, S_b and D_b might be calculated thereby a variety of different ways by referencing from $z < 0$. As S_b will not need any integer bits to be stored in memory and $D_b < 0$, immediate savings can be realized. The most common methods are briefly explained here.

Pure LUTs

LUTs can offer very good precision assuming the values of the factors are accurate enough for the operation. Compromises can be made in precision to reduce area. Because of exponential characteristic of these equations, the size of the LUTs are based on the fractional bits of the LNS and are not encoded very efficiently. This method was originally used in LNS' infancy, but it is typically only used on very small systems.

Multiplier based Interpolation

Interpolation is one of the more traditional techniques for implementing the S_b and D_b functions. Since the slope of S_b does not change dramatically, linear interpolation for addition gives satisfactory accuracy. Linear interpolation uses two tables, one for storing the values of the multiplier which are the slopes and the base values of the function [8]. For subtraction this method is not practical because a singularity exists at $z = 0$, which means slope changes significantly. Implementation of D_b becomes expensive, in terms of circuit area and power consumption, close to zero because the encoding of the slopes requires more bits.

Addition based Interpolation

Multipartite tables technique is a recent development in linear interpolation where there is no multiplication component. It is an efficient technique for a function in which the slope changes slowly. When the slope changes rapidly then more tables are needed to compensate. In this method a series of results from smaller tables, indexed by various bit portions on the input word, contribute to the computation of the final value. For the S_b function, the multipartite method generates a single table, whereas for D_b, many more separate tables are needed as the curve changes rapidly near the singularity.

The precision with a multipartite table can be higher than the previous interpolation method, but care must be taken to ensure the configuration is guarded correctly so that the error is acceptable given a limit on the hardware needed. Some times in order to achieve reasonable area it is necessary to relax accuracy in the region close to zero which causes LNS to be less accurate than FPNS. Since the accuracy varies in different applications, different degree of relaxation can be applied to the method.

The main advantage of using this method is that to the latency is reduced as there are no multipliers in the circuit [3]. Depending on the size of the table, more memory may be required compared to interpolation as the multiplier has been replaced by extra adders [3].

Real time function Calculation

Although the calculation of the S_b and D_b functions is possible in real-time, it would require some type of FPNS to generate accurate solutions. Given that the intent of the system is to avoid the overhead of FPNS, this isn't a practical solution. It is practical however to generate S_b and D_b from smaller LUTs. If the latency of such a system is comparable to the interpolation methods while still maintaining a lower area, such a system would be superior. The co-transformation method is such and will be expanded on shortly.

MDLNS Implementation

By adding multiple bases to the previous equations the classic method of LNS addition and subtraction can be extended to operate in single-digit MDLNS [1]:

$$\prod_{i=1}^{k} D_i{}^{x_i} . \prod_{i=1}^{k} D_i{}^{z_i} = \prod_{i=1}^{k} D_i{}^{x_i} + \prod_{i=1}^{k} D_i{}^{y_i}$$

$$\prod_{i=1}^{k} D_i{}^{z_i} = 1 + \prod_{i=1}^{k} D_i{}^{y_i - x_i}$$

$$\prod_{i=1}^{k} D_i{}^{x_i} . \prod_{i=1}^{k} D_i{}^{w_i} = \prod_{i=1}^{k} D_i{}^{x_i} - \prod_{i=1}^{k} D_i{}^{y_i}$$

$$\prod_{i=1}^{k} D_i{}^{w_i} = 1 - \prod_{i=1}^{k} D_i{}^{y_i - x_i}$$

As the inputs to such a table are not monotonic, it would greatly increase the complexity of calculating the table as well as encoding it efficiently. Therefore a direct MDLNS implementation is not feasible.

MDLNS Single Base Domain

To mitigate the above problem, a solution was proposed in [14] which mapped the MDLNS system into a single base domain (SBD) which is essentially a redundant LNS. This process consisted of a LUT which mapped the MDLNS exponents into a single exponent, the SBD.

$$D_1{}^{v} = \prod_{j=1}^{k} D_j{}^{a_j}$$

$$v = \sum_{j=1}^{k} a_j . \log_{D_1}(D_j)$$

Here v is a real number and for hardware implementation it is needed to be converted to integer form. This process will be done by a fixed-point representation and limited number of bits to represent the fractional part of a real number.

$$v = v_i + \frac{v_f}{m} \ \ , \ m = 2^r$$

With a single exponent, a monotonic relationship is created and a table lookup using the above method is now possible. When $z < 0$, the table values are better represented in MDLNS as the factors are always near 1. Since MDLNS is a redundant system the results of the table were also redundant so it was found they could be efficiently implemented using a Range Addressable Look-Up Table (RALUT). The result was intended to be mapped back into MDLNS using another RALUT as the SBD values were not capable of being fed-back into the input unless it was reconditioned. Although the solution offers 100% accuracy, the table sizes (in terms of bits) were not competitive with the multi-partite methods of encoding based on compatible LNS. It is important to note however that the LNS solution was not 100% correct and in some cases could be off considerably. In [14], attempts were made to try to implement the RALUTs using the multipartite approach; however this was not possible as the multipartite encoding requires a slowly changing slope and the results from the SBD tables did not meet this requirement. A recent advancement in the LNS research has yielded a new method known as the Co-Transformation which generates the subtraction results by use of the addition table as well as other smaller tables. The intent of this thesis is to use the latest incarnation of the co-transformation to further reduce table size.

<u>The Co-Transformation Method</u>

Co-Transformation is the most recent technique for performing LNS subtraction by eliminating the interpolation of D_b near the singularity. Another advantage of avoiding the singularity is to mitigate the accuracy problem of the previous approximation methods [15]. To date, four forms of the co-transformation method have been introduced: Arnold, Coleman, Improved [3], and Novel Co-Transformation [4]. Since the most recent and favorable is the Novel Co-Transformation, it will be the center of focused in this thesis. Discussion about all the mentioned methods is out of the scope and the reader can refer to references [3][4] for more details.

Novel Co-Transformation

The Novel co-transformation is based on the improved co-transformation; however it avoids some intervals, where the values become positive requiring larger LUTs as well as the compensation for special cases [4]. The novel technique uses a different function for the subtraction operation (see figure 2.3) which uses both sides of graph and combines the addition and subtraction equations, Eq. 2.3 and Eq. 2.4.

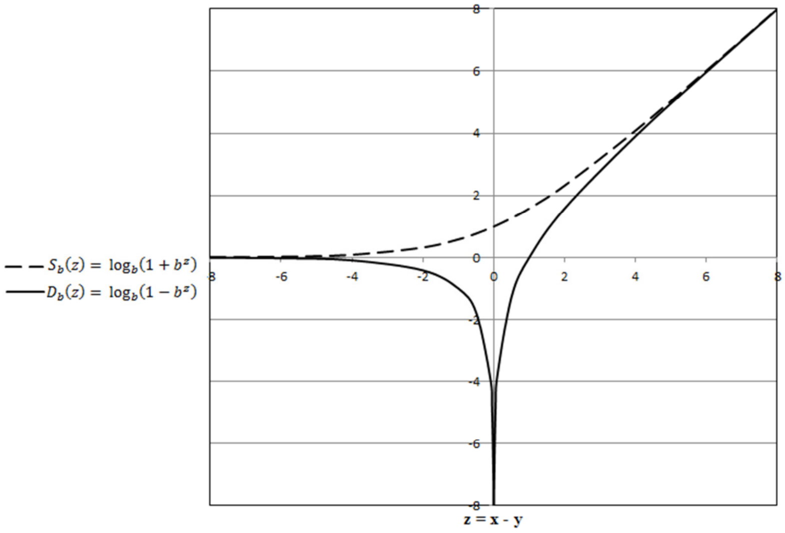

Figure 2.3: Logarithmic Addition and Subtraction Curves

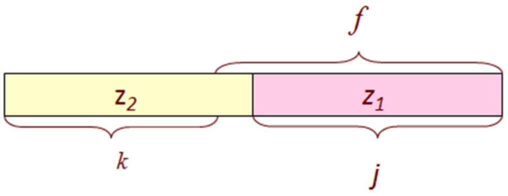

Figure 2.4: Bit partitioning of z in Novel Co-Transformation

The transformation is as follows:

$$z = z_1 + z_2$$

$$Z_1 = b^{z_1}$$

$$Z_2 = b^{z_2}$$

$$Z = Z_1 \times Z_2$$

$$Z - 1 = Z_1 \times Z_2 - 1 = (Z_1 - 1) \times \left(1 + \frac{Z_1 \times (Z_2 - 1)}{Z_1 - 1}\right)$$

$$Since\ \frac{|Z_2 - 1|}{|Z_1 - 1|} = \left|\frac{(Z_2 - 1)}{(Z_1 - 1)}\right|$$

$$Z - 1 = |Z_1 - 1| \times \left|1 + \frac{Z_1 \times |Z_2 - 1|}{|Z_1 - 1|}\right|$$

Taking the logarithm of both sides yield:

$$D_b(z) = D_b(z_1) + S_b\big(z_1 + D_b(z_2) - D_b(z_1)\big)$$

Noting that:

$$S_b(z) = z + S_b(-z)$$

$$D_b(z) = z_1 + D_b(z_2) + S_b(D_b(z_1) - z_1 - D_b(z_2)) \tag{2.5}$$

Compensating for the special cases through extra circuits is avoided by setting $D_b(0) = -2f$ in the LUTs. Calculation for $D_b(z)$ is based only on $S_b(z)$ and some smaller tables.

$$\mathrm{S_b}(z) = \begin{cases} 0 & z \leq e_{\mathrm{S}_b} \\ \mathrm{S}_b(z_H) + S_b'(z_H + \varepsilon)z_L & e_{\mathrm{S}_b} < z < 0 \\ log_b(2) & z = 0 \\ z & z > 0 \end{cases} \tag{2.6}$$

Novel co-transformation reduces complexity of circuit through decreasing area and delays of the hardware implementation [4][7][16], eliminates the special cases in improved co-transformation [4] and increases precision, but there is no benefit in terms of addition which is still implemented using the multipartite tables.

The co-transformation's inventors claim that their work unifies the most effective techniques for designing LNS units and gives a more complete practical study of the design space than any previous works [3]. The intent of this work is to combine the idea of co-transformation with MDLNS to try to reduce the table sizes from the only known method available.

Summary

So far in this thesis, it has been explained that depending on the ratio of multiplication over addition and subtraction operations in a system, sometimes LNS representation is a better choice. It has some problems in terms of implementation especially for subtraction near the singularity but studies have shown improvements in implementation depending on S_b and D_b. Based on LNS, another concept has been introduced by adding multiple bases associated with range of exponents called MDLNS. Different techniques have been developed to overcome LNS implementation issues. Co-Transformation and specifically Novel Co-transformation recently tried to eliminate LNS subtraction problem near the singularity and increase the accuracy of these operations. In the following chapters this new method will be applied to MDLNS and results will be compared with previous works.

CHAPTER III

DESIGN AND METHODOLOGY

Introduction

This chapter will discuss an overview of the proposed algorithm as the low-level coding itself is very specific to the host system.

Proposed Algorithm

This proposed algorithm is based on using the Novel Co-Transformation with the SBD model to implement both addition and subtraction for the MDLNS. MATLAB is the host language for which the software was written. The algorithm performs a brute force method of searching for the best parameter which result the minimum implementation area (size of LUTs). The algorithm is shown before in a brief pseudo code format. The full MATLAB code is available in Appendix A. In includes vector optimizations to further increase the performance.

```
Generate core MDLNS sequence with real SBD values, $seqnum$ rows
Calculate integer bits, $k$
For $r = 2$ to …
Generate integer SBD values in tables based on $m = 2^r$
Set $f = r$
For $p = f - dp$ to $f$
Set precision of all tables and arithmetic to $p$ fractional bits
For $q = 0$ to $f$
Generate $S_b$ with $q$ being the number of bits used for multiplication with the slope
```

```
For $x2 = 0$ to $2^k \times seqnum$
    For $x1 = x2$ to $2^k \times seqnum$
        Find real solution for $ordinal(x1) + ordinal(x2)$
        Find difference in SBD values of $ordinal(x2) - ordinal(x1)$
        Lookup value in $S_b$ (input is negative)
        Add to largest value
        Find difference between approximation and true MDLNS value
        Add to error count if necessary
    End $x1$
End $x2$
For $j = 1$ to $f - 1$
    Generate $D_b$ smaller tables
    For $x2 = 2^k \times seqnum$ to 2
        For $x1 = x2 - 1$ to 1
            Find real solution for $ordinal(x2) - ordinal(x1)$
            Find difference in SBD values of $ordinal(x2) - ordinal(x1)$
            Break up work into $z1$ and $z2$
            Lookup values in smaller $D_b$ LUTs and calculate offset (use $S_b$ as well)
            Add to smallest value
            Find difference between approximation and true MDLNS value
            Add to error count if necessary
        End $x1$
    End $x2$
```

```
Save error values to table
Record new lowest error
End $j$
End $q$
End $p$
If error reached minimum, end $r$ loop
End $r$
Sort results my least error
Return
```

Brief Explanation

The algorithm begins by generating the core MDLNS sequence [18] along with the SBD mapping in a real form. The number of elements is $seqnum$ and it depends on the number of bases and the range on each base; this value can become larger quickly if there are more than 2 bases.

The number of integer bits is then calculated using the method in [14]; this value will affect the LUTs greatly as each additional bit doubles their size.

The main loop then begins cycling through r starting from 2 in order to complete the SBD integer form (v) such that there is no overlap in the sequence, that is no duplicate entries.

In order to find the smallest tables, the algorithm next cycles through all the generation parameters. f is set to r as there is no reason to allocate fewer of more bits to it. For each f, p cycles from $f - deltap$ to f to explore the effects of various bit precisions on the LUT sizes. For each p, q is also cycled to explore the effects of interpolation of S_b on the results (see Eq. 2.6). This completes the three nested loops for calculating almost all the possible parameters for the addition and subtraction LUTs. In this nested loop, the error associate with the addition and subtraction tables is calculated and the best configuration is selected.

For addition, the S_b LUTs are generated using the formula in Eq. 2.6. These tables are verified by adding all possible MDLNS values with each other using the method found in [14]. Since the operation is based on the relative difference between two numbers, any power of 2 scaling applied to the two numbers will result in the same answer scaled by the same value. For example, computing 1+2=3 is the same as 2+4=6, etc. This considerably reduces the number of possible combinations so that the whole table can be verified in a finite amount of time. After the completion of $x1$ loop, the running error is evaluated to see if it is far beyond the best or beyond the minimum allowed, and if so, the $x2$ loop is also terminated and the subtraction tables are skipped. This helps improve the performance of the optimization.

A similar operation is used for verifying the D_b LUTs. Here, j is cycled from 1 up to $f-1$ as j only affects the subtraction tables. The tables are first generated using Eq. 2.5 and Eq. 2.6 and a dual nested loop with $x2$ and $x1$ are configured such that one value is always larger than the other to avoid sign issues. The same scaling optimizations apply such that, for example, 2-1=1 is evaluated and 4-2=2 is not. The $x1$ loop is also monitored to stop if excessive error is reach to further improve running speed.

After each table verification is complete, the parameters, the table sizes and errors are recorded into a running list. Each entries error is compared with a running error to monitor if the minimum error has been reached.

After the completion of the j, q and p loops, the running list is sorted and any entries that exceed the best error by a certain factor are removed to conserve memory.

If the target minimum error has not been achieved, r is increased and the loop continues. If the minimum has been met, the running list is sorted by 3 keys: minimum error, minimum overall bit size and minimum implementation bits sizes. The data is then returned to the calling function.

Optimizations

There are a number of optimizations included in the software code which are not discussed in the above algorithm as they are out of the scope of this thesis. However, a few techniques will be mention here as to prepare the reader for interpreting the code in Appendix A.

1. All static computational values are cached into tables so that expensive log, exp, and other function are minimized to only a small portion of overall run-time. This can require more memory, but the speed gains are worth the sacrifice.

2. Any arrays or matrices are pre-allocated before use as this can have a significant impact on performance. During earlier runs of the software, virtual most of the computing time was simply memory management instead of data processing.

3. The function is programmed as such as MATLAB performs further optimizations in run-time as compared to a script

4. Vector and matrix processing is heavily used to increase performance greatly. MATLAB, as a programming language, is not very fast. Using loops and single value functions is easily out performed by other languages such as C. Where MATLAB really performs well is in vector and matrix manipulation. Every opportunity is made to make

use of this as MATLAB parallelizes the code run-time to work on multiple threads and processors. On the Canadian computational cloud “Sharcnet” or “Compute-Canada”, this code was observed to operate across over 30 CPUs during large vector and matrix operations; a significant performance improvement.

Summary

This chapter briefly explained the proposed algorithm of implementing both addition and subtraction for the MDLNS with using Novel Co-Transformation along with SBD model. Step by Step Explanation of the MATLAB Code is discussed in this chapter and the code can be found in Appendix A.

The goal of this algorithm is to find the best combinations of all possible parameters which result the minimum implementation size of the LUTs and also minimum error associated with the addition and subtraction tables. Furthermore, some optimization techniques have been used to maximize the performance of the software to arrive at results faster.

CHAPTER IV

ANALYSIS OF RESULTS

Introduction

This chapter presents the results of running numerous simulations for weeks at a time. Even though a significant amount of code optimization was applied to improve performance in the MATLAB environment, the computation running times were long and only a small portion of data could be generated to meet the thesis deadlines.

Single Base Results

The following results are generated from using a single non-binary base of 3. The range on the exponents has a full swing from positive to negative. Table 4.1 summaries the three sets of results (no error, 1 unit error in addition or subtraction, and 1 unit error in addition and subtraction) compared to the previously known RALUT system. A full implementation analysis of each scenario would have required much more time, more coding, and the results would have only been applicable to a particular technology. To simplify matters, a general area scaling was performed using data from custom layouts [19] where each RALUT and LUT address decoder is 14 and 4 times larger than an output bit respectively. This area scaling value, although not 100% accurate, can give some indication as to the size of the system. The table rows for the proposed method include only the rows using from the S_b and D_b tables and not the full range, although that information can be extracted from the parameters.

Table 4.1: Single Base Results

		RALUT				Proposed					Proposed with Error	
Bases	Range	*r*	Add Table (Rows, Input Bits, Output Bits)	Sub Table (Rows, Input Bits, Output Bits)	Scaled Area with No Error	*r,k,j,q,p*	S_b (Rows, Input Bits, Output Bits)	D_{b1} (Rows, Input Bits, Output Bits)	D_{b2} (Rows, Input Bits, Output Bits)	Scaled Area with No Error (Relative to RALUT)	Scaled Area with 1 Unit Error in Add **or** Sub (Relative to RALUT)	Scaled Area with 1 Unit Error in Add **and** Sub (Relative to RALUT)
3	1	4	7x8x6	10x8x8	2026	6,3,5,6,5	8x3x5	5x5x5	16x1x5	597 (29%)	548 (27%)	470 (23%)
3	2	5	13x9x7	17x9x9	4024	8,3,1,7,8	16x4x8	1x1x8	69x7x8	3708 (92%)	2033 (50%)	1880 (46%)
3	3	5	18x9x7	26x9x9	5904	9,3,8,6,7	64x6x7	19x8x7	16x1x7	3093 (52%)	4334 (73%)	688 (11%)
3	4	7	24x11x9	35x11x11	9687	10,3,1,3,9	135x10x9	2x1x9	132x9x9	14165 (146%)	13145 (135%)	832 (8%)
3	5	8	31x12x10	41x12x12	12898	10,3,1,3,9	167x10x9	2x1x9	164x9x9	17557 (136%)	15825 (122%)	832 (6%)
3	6	8	37x12x10	51x12x12	15766	12,3,3,9,12	64x6x12	2x3x12	195x9x12	14052 (89%)	10881 (69%)	3407 (21%)
3	7	7	42x11x9	71x11x11	18561	12,3,2,9,12	64x6x12	1x2x12	228x10x12	16916 (91%)	11076 (59%)	3252 (17%)
3	8	7	40x11x9	64x11x11	17080	12,3,2,8,12	127x7x12	1x2x12	260x10x12	21740 (127%)	11652 (68%)	3396 (19%)
3	9	7	53x11x9	89x11x11	23324	12,3,2,4,12	292x11x12	1x2x12	291x10x12	34996 (150%)	30060 (128%)	3540 (15%)
3	10	8	54x12x10	86x12x12	25092	13,3,3,8,13	237x8x13	1x3x13	323x10x13	31685 (126%)	20884 (83%)	3684 (14%)
3	11	8	57x12x10	96x12x12	27426	15,3,1,11,15	127x7x15	2x1x15	497x14x15	46750 (170%)	17526 (63%)	3828 (13%)
3	12	9	66x13x11	110x13x13	34188	15,3,1,11,15	128x7x15	2x1x15	542x14x15	50528 (147%)	18291 (53%)	4196 (12%)
3	13	10	70x14x12	115x14x14	38710	15,3,1,11,15	128x7x15	2x1x15	582x14x15	53848 (139%)	18336 (47%)	4340 (11%)
3	14	10	60x14x12	104x14x14	34320	15,3,1,11,15	128x7x15	2x1x15	620x14x15	57002 (166%)	30749 (89%)	4484 (13%)
3	15	10	81x14x12	124x14x14	42888	15,3,1,11,15	128x7x15	2x1x15	662x14x15	60488 (141%)	31718 (73%)	4628 (10%)
3	16	10	88x14x12	130x14x14	45604	16,3,2,12,15	128x7x15	4x2x15	590x14x15	54566 (119%)	32738 (71%)	4772 (10%)
3	17	10	95x14x12	136x14x14	48320	17,3,3,13,15	128x7x15	8x3x15	577x14x15	53611 (110%)	33095 (68%)	4916 (10%)
3	18	10	100x14x12	143x14x14	50830	20,3,1,16,18	128x7x18	2x1x18	813x19x18	92110 (181%)	38127 (75%)	5060 (9%)
3	19	10	110x14x12	149x14x14	54170	20,3,1,16,18	128x7x18	2x1x18	862x19x18	97304 (179%)	38519 (71%)	5204 (9%)
3	20	9	104x13x11	150x13x13	49322	20,3,1,16,18	128x7x18	2x1x18	901x19x18	101438 (205%)	39254 (79%)	5348 (10%)
3	21	10	124x14x12	161x14x14	59602	20,4,12,16,18	256x8x18	161x12x18	1354x8x18	112790 (189%)	70245 (117%)	8564 (14%)
3	22	10	129x14x12	163x14x14	61062	20,4,12,16,18	256x8x18	167x12x18	1418x8x18	117410 (192%)	71766 (117%)	8708 (14%)
3	23	10	134x14x12	173x14x14	64202	20,4,12,16,18	256x8x18	175x12x18	1481x8x18	122096 (190%)	73899 (115%)	8852 (13%)
3	24	10	136x14x12	180x14x14	66088	20,4,12,16,18	256x8x18	183x12x18	1546x8x18	126914 (192%)	74322 (112%)	8996 (13%)

Upon examining the results, the proposed method is no more than twice the size of the results from the RALUT. This can be expected due to the fact that the RALUTs can compress a large amount of data scattered across many rows into a single one. [14] shows that the MDLNS addition and subtraction LUTs are very large prior to being implemented in RALUTs. Once a single unit error is allowed in either addition or subtraction, the tables are smaller in most cases. An error in both addition and subtraction result in much smaller tables, as much as 6% the size of the RALUT. These conditions are more significant as the S_b table in a LNS system is expected to have error in it; no implementation has zero error. In fact, the S_b table in LNS can have a number of solutions which provide up to a single unit error. Once the tables in LNS are implemented into a multipartite circuit, further errors are incurred [14], however they are deemed acceptable as they are a compromise for large savings in circuit area. The same savings is expected to happen here further, however only a small portion of the S_b tables are actually used and the multipartite system is constructed to generate a complete table. By including the non-used values in the generation phase, the LUT size will be much larger and consume more area. If it were designed to output only these used values, the parameters for generation would be far more relaxed and the LUTs would be much smaller and use far less area. This feature does not currently exist so modifications need to be made to the multipartite system to allow the implementation of sparse tables, which is not trivial as the smaller LUTs are based on the complete input map.

The choice of r for the proposed method is clearly larger than that of the RALUT. This implies that there may be some potential for selecting the same r as in the RALUT method while still achieving zero error. This will probably require some time of modification of the tables and re-verification to ensure a 100% no error system.

Additionally, the selection of the non-binary base of 3 could have inflated these results just as other arbitrary bases could have easily reduced them. [8] Shows how selecting optimal bases can significantly impact the implementation size of a digital filter.

Two base Results

The following results are generated from using two non-binary bases of 3 and 5. The range on the exponents has a full swing from positive to negative for both bases, so the effective complexity of the system increases exponentially as compared to the single base systems. For example, in the single base system, a range of -10 to 10 would result in 21 (-low + high +1) components in the core MDLNS sequence. For a two base system with a range of -10 to 10 on each base, the resulting system would have 21x21 or 441 core components. Table 4.2 summaries the three sets of results (no error, 1 unit error in addition or subtraction, and 1 unit error in addition and subtraction) compared to the previously known RALUT system. The same general area scaling rule was applied to obtain reasonable results.

A similar trend is noticed here compared to single base results; the error free systems are larger than the original RALUT system, but not usually by more than 3 times. Once error is allowed, a significant savings can be seen. This reiterates the need to further examine the potential for further table reduction. At this point, the resulting tables have not been inspected to determine if further trial methods can be utilized (interpolation, etc.).

Table 4.2: Two Base Results

		RALUT				Proposed					Proposed with Error	
Bases	Range	*r*	Add Table (Rows, Input Bits, Output Bits)	Sub Table (Rows, Input Bits, Output Bits)	Scaled Area with No Error	*r,k,j,q,p*	S_b (Rows, Input Bits, Output Bits)	D_{b1} (Rows, Input Bits, Output Bits)	D_{b2} (Rows, Input Bits, Output Bits)	Scaled Area with No Error (Relative to RALUT)	Scaled Area with 1 Unit Error in Add **or** Sub (Relative to RALUT)	Scaled Area with 1 Unit Error in Add **and** Sub (Relative to
3,5	1	7	23x11x9	41x11x11	10514	21,3,1,19,19	32x5x19	2x1x19	192x20x19	22606 (215%)	11689 (111%)	766 (7%)
3,5	2	10	112x14x12	156x14x14	56056	21,4,15,18,19	128x7x19	131x15x19	939x6x19	71766 (128%)	30388 (54%)	16206 (28%)
3,5	3	12	228x17x14	311x17x17	136761	26,4,20,20,24	1024x10x24	283x20x24	1021x6x24	160312 (117%)	129696 (94%)	19146 (13%)
3,5	4	13	382x18x15	546x18x18	249414	26,4,22,19,25	2048x11x25	493x22x25	256x4x25	211613 (84%)	574714 (230%)	42773 (17%)
3,5	5	13	613x18x15	865x18x18	397221	26,4,8,19,24	2047x11x24	256x8x24	7042x18x24	942236 (237%)	424829 (106%)	105430 (26%)
3,5	6	13	842x18x15	1204x18x18	549894	26,4,8,19,24	2048x11x24	256x8x24	9973x18x24	1270576 (231%)	857278 (155%)	120353 (21%)
3,5	7	13	1107x18x15	1623x18x18	733779	26,4,23,19,24	2048x11x24	1567x23x24	128x3x24	327692 (44%)	519916 (70%)	132743 (18%)
3,5	8	19	1657x24x21	2290x24x24	1415949	26,4,4,18,25	4096x12x25	16x4x25	18395x22x25	2672619 (188%)	976640 (68%)	255683 (18%)
3,5	9	21	2078x26x23	2910x26x26	1939086	26,4,1,18,25	4096x12x25	2x1x25	31808x25x25	4783994 (246%)	1350204 (69%)	284543 (14%)
3,5	10	21	2353x26x23	3383x26x26	2229981	26,4,1,18,25	4096x12x25	2x1x25	39414x25x25	5856440 (262%)	1371220 (61%)	454886 (20%)
3,5	11	21	2979x26x23	4338x26x26	2844693	26,4,1,18,25	4096x12x25	2x1x25	47362x25x25	6977108 (245%)	1088544 (38%)	479042 (16%)
3,5	12	22	3523x27x24	5156x27x27	3504426	26,4,1,18,25	4096x12x25	2x1x25	56262x25x25	8232008 (234%)	3416674 (97%)	577938 (16%)

Summary

Results for a single (3) and two (3, 5) non-binary base systems were shown to have a slightly larger scaled area than the original RALUT implementation. However, once a single unit error was allowed, the scaled area dropped significantly especially in the cases where it was allowed on both addition and subtraction. These scaled values have yet to be fully optimized as the multipartite tables cannot be applied since the tables are sparse and incomplete. This will be a task for another researcher in the future.

CHAPTER V

CONCLUSIONS AND RECOMMENDATIONS

Conclusions

This goal of this thesis was to improve the implementation of addition and subtraction circuits in MDLNS based on earlier works which were applied to LNS only. The Novel Co-transformation method for subtraction in LNS was analysed and successfully applied to the MDLNS, which is a super-set of the LNS. This resulted in the development of a programmable framework for testing various bases and exponent ranges to investigate the method's performance. The resulting tables show very good promise when a certain level of error is allowed, but for zero error systems, more optimizations still need to be performed to obtain solid results. The choice of m, or 2^r, appears to be increasing at a larger rate than in the previous RALUT method. It may be possible to adjust the tables during verification to select smaller parameters and therefore smaller tables.

Although the software code is written in MATLAB to ease development time(with many optimizations to improve run-time performance), the execution times are still quite high and limit the analysis on systems with more than one non-binary base and larger exponent ranges.

Lastly, the selection of bases 2, 3, 5, 7, etc. is historical as it provides true orthogonal bases, but it is possible that better results can be obtained from a more optimal set of bases [8].

Future Work

Unfortunately, the resulting low table utilization introduces a great degree of sparseness in the tables. The existing multipartite method for efficient table implementation cannot be applied as the resulting hardware will target all outputs as opposed to just that small amount which is actually used. This would result in larger tables than necessary. This change is recommended to be investigated by another researcher in the future.

Ultimately, a full implementation will indicate which method is the best. This will require the above multipartite implementation, the circuit to perform the addition and subtraction operation, as well as the associated interconnecting circuits. All of this would be synthesised and compared with current technologies to see which method is best.

The software could be recoded in a higher performance language (C, for example) to better manage memory and resources while decreasing execution time.

Execution times could be further improved by examining the results from many scenarios to see what the trends of the parameters are. This software performs a brute force approach (trying all possible combinations), but it may not be necessary if statistical data suggests certain combinations are either favourable or unlikely to give good results.

APPENDICES

APPENDIX A

Software

```
function[ResultMinErr]=mdlnscotrans(base,expl,exph,startm,stopm,minerro
r,maxrounds)
format short g

l2=log(2);
b=2;
lb=log(b);
vi=0;

NRows = 1;
k = size(base,2);
MaxF = 100;

TBArea = -1;
TBErr = -1;
TBErrArea = 1e99;
TBErrZero=0;
%tic;

ind_r=1;
ind_k=ind_r+1;
ind_j=ind_k+1;
ind_q=ind_j+1;
ind_p=ind_q+1;
ind_ar=ind_p+1;
ind_ae=ind_ar+1;
ind_sr1=ind_ae+1;
ind_sr2=ind_sr1+1;
ind_se=ind_sr2+1;
ind_tr=ind_se+1;
ind_tf=ind_tr+1;
ind_te=ind_tf+1;
ind_end=ind_te;

deltap=2;
deltaq=2;

ErrorFactor=2;

NRows = 1;
for tk=1 : k
    NRows = NRows * (exph(tk)-expl(tk)+1);
end
A=zeros(NRows+1,k+4);
tempc=1;
for tk=1:k
    n=expl(tk);
    if tk == 1
       tempc=1;
    else
       tempc = tempc * (exph(tk-1)-expl(tk-1)+1);
    end
    for h=1:NRows
        A(h,tk+1) = n;
        R = rem(h,tempc);
        if R == 0
           if n<exph(tk)
              n=n+1;
           elseif n==exph(tk)
              n=expl(tk);
           end
        end
    end
end
lbase=log(base)';
for h=1:(NRows)
    res = exp((A(h,2:k+1) * lbase));
    [x1,x2]=log2(res);
    x1=x1*2;
    x2=x2-1;
    A(h,1)=x2;
    A(h,k+2) = x1;
    A(h,k+3) = log(x1)/lb;
end
clear lbase
A(NRows+1,k+2)=2^(vi+1);
A=sortrows(A,k+2);
A(NRows+1,:)=A(1,:);
A(NRows+1,1)=A(1,1)+1;
A(NRows+1,k+2)=2^(vi+1);
A(NRows+1,k+3)=A(1,k+3)+1;
A(NRows+2,:)=A(2,:);
A(NRows+2,1)=A(2,1)+1;
A(NRows+2,k+2)=A(2,k+2)*2;
A(NRows+2,k+3)=A(2,k+3)+1;

u1 = 100;
for l=1:(NRows-1)
    divr = A(l+1,k+2)/A(l,k+2);
    if divr<u1
        u1=divr;
    end
end

A

numberofintegerbits1 = ceil(log((log(2/(u1-1))/l2)*110/100)/l2);
numberofintegerbits2 = ceil(log((log(2/(1-(1/u1)))/l2)*110/100)/l2);
ik = numberofintegerbits1;
Mvi=2^ik;
disp(sprintf('Number of Integer Bits=%d',ik));

Rownum=1;
TempAcc = ones(100,ind_end)*1e15;

for r=startm:stopm;
```

```
m=2^r;
disp(sprintf('m=%d',m));
f=r;

u1=0;
for h=1:(NRows)
    A(h,k+4) = round(A(h,k+3)*m);
    if (h>1 && A(h,k+4)<=A(h-1,k+4))
        disp('Overlap in mapping, usng next "m".');
        u1=-100;
        break;
    end
end
    if (u1<-1)
    continue;
end

A(NRows+1,k+4)=A(1,k+4)+m;
A(NRows+2,k+4)=A(2,k+4)+m;

A

% Cache recurring computations

y_a=zeros(1,NRows*Mvi);
z_a=zeros(1,NRows*Mvi);

for x1=1:(NRows*Mvi)
    NCRow1=mod(x1-1,NRows)+1;
    y_a(x1)=A(NCRow1,k+2)*(2^(floor((x1-1)/NRows)));
    z_a(x1)=floor((x1-1)/NRows)+(A(NCRow1,k+4)/m);
end

ADDPQJ=ones(deltap+1,f+ik,f-1)*-1;
SUBPQJ=ones(deltap+1,f+ik,f-1)*-1;

SUBPQJerrtot=zeros(deltap+1,f+ik,f-1);
SUBPQJerrnum=zeros(deltap+1,f+ik,f-1);
ADDPQJerrtot=zeros(deltap+1,f+ik,f-1);
ADDPQJerrnum=zeros(deltap+1,f+ik,f-1);

PrevLocalTBErr = 1e99;
mbreak = 0;

% Create fast searching cache
x2=1;
x3=1.0;
fastmap=zeros(1,m,'double');
for x1=1:1:m
    while (x3<A(x2,k+2) || x3>=A(x2+1,k+2))
        x2=x2+1;
    end
    fastmap(x1)=x2;
    x3=x3+1/m;
end
```

```
    % Create fast nearest cache
    x2=1;
    fastnear=zeros(1,m,'double');
    for x1=1:1:NRows
        x3=round(log((A(x1,k+2)+A(x1+1,k+2))/2)/lb*m);
        while x2<=x3
            fastnear(x2)=x1;
            x2=x2+1;
        end
    end
    x1=x1+1;
    while x2<=m
        fastnear(x2)=x1;
        x2=x2+1;
    end


    for f=r:1:r;

        LocalTBErr=-1;
        fp2=2^f;



        for p=f-deltap:1:f
            pp2=2^p;
            ip = p-f+deltap+1;

            for q=0:1:f;
            iq = q+1;

            qskip=0;
            qbreak=0;

            j=0;
            disp(sprintf('r=%d, j=%d, q=%d, p=%d, TBErr=%f,
TBErrArea=%f',r,j,q,p,TBErr,TBErrArea));
            worst=0;

            clear z_l_a
            clear td_b1_a
            clear td_b1_a_hit
            clear td_b2_a
            clear td_b2_a_hit

            sbf=f;
            sbk=ik;
            sbj=q;
            sbp=p;

            sbfp2=2^sbf;
            sbjp2=2^sbj;
            sbpp2=2^sbp;
            sbi=2^(sbf-sbj);
                        sbz_h=-[0:1:2^(sbf-sbj+sbk)+1]/sbi;
            sbts_b            =            round((log(1+(ones(1,2^(sbf-
sbj+sbk)+2)*b).^sbz_h)/lb)*sbpp2)/sbpp2;
                        clear sbz_h
            sbts_b_hit=zeros(1,2^(sbf-sbj+sbk)+2,'double');
            for x2=1:(NRows*Mvi)
                y2=y_a(x2);
                z2=z_a(x2);

                for x1=x2:(NRows*Mvi)
                    y1=y_a(x1);
                    z1=z_a(x1);
                    in=z2-z1;
if in<-2^sbk
    s_b = 0 ;
elseif in>0
    s_b = in;

else
    i=floor(-in*sbi)+1;
    sbtsb=sbts_b(i);
    sbts_b_hit(i)=1;

    s_b=sbtsb+(sbts_b(i+1)-sbtsb)*sbi*mod(floor(-
in*sbfp2+0.5),sbjp2)/sbfp2;
end
                    approx=floor((z1+s_b)*m+0.5)/m;
                    fn_i=fastnear( mod(approx*m,m)+1 );
                    fn_e=floor(approx);
                    cor=y1+y2;
                    [cor_m,cor_e]=log2(cor);
                    cor_m=cor_m*2;
                    cor_e=cor_e-1;
                    fm1=double(fastmap(floor((cor_m-1)*m+1)));
                    while (cor_m>=A(fm1+1,k+2))
                        fm1=fm1+1;
                    end
                    cor_il=fm1;
                    cor_ih=fm1+1;
                    cor_eh=(A(cor_ih,k+2)-cor_m);
                    cor_el=(cor_m-A(cor_il,k+2));
                    cor_slack=0;
                    cor_i=cor_il;
                    % Check if error is split between both entries
                    if abs(abs(cor_eh-cor_el)/cor_eh)<0.001
                        cor_slack=1;
                    elseif cor_eh<cor_el
                        cor_i=cor_ih;
                    end
                    cor_o=cor_e*NRows+cor_i;
                    fn_o=fn_e*NRows+fn_i;
                    err=0;
                    if fn_o<cor_o
                        err=cor_o-fn_o;
                    end
                    if fn_o>cor_o+cor_slack
                        err=fn_o-cor_o-cor_slack;
                    end
                    if (err>0)

ADDPQJerrtot(ip,iq,:)=ADDPQJerrtot(ip,iq,1)+err;
                        ADDPQJerrnum(ip,iq,:)=ADDPQJerrnum(ip,iq,1)+1;
                    end
                    worst=max(err,worst);
                end


err=ADDPQJerrtot(ip,iq,1)/(ADDPQJerrnum(ip,iq,1)+(ADDPQJerrnum(ip,iq,1)
==0));
                if ( err >PrevLocalTBErr*ErrorFactor) || (err>minerror)
                    disp('Stopping    internal    calculation    due    to
excessive error');
                    ADDPQJerrtot(ip,iq,1)=1e90;
                    ADDPQJerrnum(ip,iq,1)=1;
                    qskip=1;
                    break
                end


            end
            ADDPQJ(ip,iq,:)=worst;

            if (qskip>0)
                continue;
            end

            for j=1:f-1;

            disp(sprintf('r=%d,    j=%d,    q=%d,    p=%d,    TBErr=%f,
TBErrArea=%f',r,j,q,p,TBErr,TBErrArea));
            jp2=2^j;
            jskip=0;

            TempAcc(Rownum, ind_r) = r;
            TempAcc(Rownum, ind_k) = ik;
            TempAcc(Rownum, ind_j) = j;
            TempAcc(Rownum, ind_q) = q;
            TempAcc(Rownum, ind_p) = p;
            TempAcc(Rownum,                  ind_ae)                =
ADDPQJerrtot(ip,iq,j)/(ADDPQJerrnum(ip,iq,j)+(ADDPQJerrnum(ip,iq,j)==0)
);
            TempAcc(Rownum, ind_se) = 0;
            TempAcc(Rownum, ind_tf) = 2^(sbf-sbj+sbk) + 2^(f+ik-j)+2^j;
            TempAcc(Rownum, ind_te) = TempAcc(Rownum, ind_ae);


            worst=0;
            omega = -2*f;
            td_b1_a_hit=zeros(1,jp2,'double');
            z_l_a=[ log(1-b^(omega))/lb [1:1:jp2-1]/fp2 ];
                        td_b1_a      =      round((log(abs(ones(1,jp2)-
b.^z_l_a))/lb)*pp2)/pp2;
            td_b2_a_hit=zeros(1,2^(f+ik-j),'double');
            fjp2=2^(f-j);
            z_h_a=[ log(1-b^(omega))/lb [1:1:2^(f+ik-j)-1]/fjp2 ];
                        td_b2_a    =   round((log(abs(ones(1,2^(f+ik-j))-
b.^z_h_a))/lb)*pp2)/pp2;
            clear z_h_a

            for x2=(NRows*Mvi):-1:2
                y2=y_a(x2);
                z2=z_a(x2);

                for x1=x2-1:-1:1
                    y1=y_a(x1);
                    z1=z_a(x1);

                    cor=y2-y1;
                    NZ=z2-z1;
                        z_i = mod(NZ*fp2,jp2)+1;
                        z_l = z_l_a(z_i);
                        td_b1 = td_b1_a(z_i);
                        td_b1_a_hit(z_i)=1;
                        td_b2 = td_b2_a(floor(NZ*fjp2)+1);
                        td_b2_a_hit(floor(NZ*fjp2)+1)=1;
                        in=td_b1-z_l-td_b2;

if in<-2^sbk
    s_b = 0 ;
elseif in>0
    s_b = in;
else
    i=floor(-in*sbi)+1;
    sbtsb=sbts_b(i);
    sbts_b_hit(i)=1;

    s_b=sbtsb+(sbts_b(i+1)-sbtsb)*sbi*mod(floor(-
in*sbfp2+0.5),sbjp2)/sbfp2;
end
                        approx=floor((z1+z_l+td_b2+s_b)*m+0.5)/m;


                    err=0;

                        fn_i=fastnear( mod(approx*m,m)+1 );
                        fn_e=floor(approx);
                        [cor_m,cor_e]=log2(cor);
                        cor_m=cor_m*2;
                        cor_e=cor_e-1;
                        if cor_m<1
                            cor_m=cor_m*2;
                            cor_e=cor_e-1;
                        end
                        fm1=double(fastmap(floor((cor_m-1)*m+1)));
                        while (cor_m>=A(fm1+1,k+2))
                            fm1=fm1+1;
                        end
                        cor_il=fm1;
                        cor_ih=fm1+1;
                        cor_eh=(A(cor_ih,k+2)-cor_m);
                        cor_el=(cor_m-A(cor_il,k+2));
                        cor_slack=0;
                        cor_i=cor_il;
                        if cor_eh<cor_el
                            cor_i=cor_ih;
                            else
                        % Check if error is split between both entries
                            if abs(abs(cor_eh-cor_el)/cor_eh)<0.001
                                cor_slack=1;
                            end
                        end
                        cor_o=cor_e*NRows+cor_i;
                        fn_o=fn_e*NRows+fn_i;
                        err=0;
                        if fn_o<cor_o
                            err=cor_o-fn_o;
                        end
                        if fn_o>cor_o+cor_slack
                            err=fn_o-cor_o-cor_slack;
                        end

                        if (err>0)

SUBPQJerrtot(ip,iq,j)=SUBPQJerrtot(ip,iq,j)+err;

SUBPQJerrnum(ip,iq,j)=SUBPQJerrnum(ip,iq,j)+1;
                        end
                        worst=max(err,worst);

                end
                TempAcc(Rownum, ind_se) =
SUBPQJerrtot(ip,iq,j)/(SUBPQJerrnum(ip,iq,j)+(SUBPQJerrnum(ip,iq,j)==0)
);
                TempAcc(Rownum, ind_te) = sqrt(TempAcc(Rownum,
ind_ae)^2 + TempAcc(Rownum, ind_se)^2);
                if TempAcc(Rownum, ind_te)>PrevLocalTBErr*ErrorFactor
                    disp('Stopping internal calculation due to
excessive error');
                    TempAcc(Rownum, ind_se)=1e90;
                    jskip=1;
                    break
                end

            end
            SUBPQJ(ip,iq,j) = worst;

            if (jskip>0)
                break;
            end
            TempAcc(Rownum, ind_te) = sqrt(TempAcc(Rownum, ind_ae)^2 +
TempAcc(Rownum, ind_se)^2);
            sbts_b_hit(2^(sbf-sbj+sbk)+1)=0;
            TempAcc(Rownum, ind_ar) = sum(sbts_b_hit);
            TempAcc(Rownum, ind_sr1) = sum(td_b1_a_hit);
            TempAcc(Rownum, ind_sr2) = sum(td_b2_a_hit);
            TempAcc(Rownum, ind_tr) = TempAcc(Rownum, ind_ar) +
TempAcc(Rownum, ind_sr1) + TempAcc(Rownum, ind_sr2);
```

```
            Rownum = Rownum +1;

            if (TBErr < 0)
                TBErr = TempAcc(Rownum-1, ind_te);
                if (TBErr <= minerror)
                    TBErrArea = TempAcc(Rownum-1, ind_tr);
                end
            end
            if (TBErr > TempAcc(Rownum-1, ind_te))
                TBErr = TempAcc(Rownum-1, ind_te);
                if (TBErr <= minerror)
                    TBErrArea = TempAcc(Rownum-1, ind_tr);
                end
            elseif TBErr == TempAcc(Rownum-1, ind_te) && TBErr <=
minerror
                TBErrArea = min(TBErrArea,TempAcc(Rownum-1, ind_tr));
            end

            if (q==0 && TBErr > minerror)
                disp('No point, skipping to next p');
                qbreak=1;
                break;
            end

            if (LocalTBErr < 0)
                LocalTBErr = TempAcc(Rownum-1, ind_te);
            end
            if (LocalTBErr >= TempAcc(Rownum-1, ind_te))
                LocalTBErr = TempAcc(Rownum-1, ind_te);
            end

        end %j

        if (qbreak>0)
            break;
        end

        end %q
        end %p

        if (LocalTBErr < PrevLocalTBErr)
            PrevLocalTBErr = LocalTBErr;
        else
            disp(sprintf('Stopping f=%d.',f));
            break;
        end

    end %f

    TempAcc=sortrows(TempAcc,ind_tr);
    TempAcc=sortrows(TempAcc,ind_tf);
    TempAcc=sortrows(TempAcc,ind_te);
    j=find(TempAcc(:,ind_te)>0,1,'first');
    f=find(TempAcc([j:1:Rownum-
1],ind_te)>TempAcc(j,ind_te)*ErrorFactor,1,'first');
    if (size(f,1)>0)
        TempAcc=TempAcc(1:1:j+f-2,:);
        Rownum=size(TempAcc,1)+1;
    else
        TempAcc=TempAcc(1:1:Rownum-1,:);
    end
    ResultMinErr = TempAcc;

    if (TBErr>=0 && TBErr <= minerror)
        TBErrZero=TBErrZero+1;
        if (TBErrZero >= maxrounds)
            disp(sprintf('Stopping mp=%d. error zero for past %d
rounds.',r,maxrounds));
            break;
        end
    end

    if (mbreak>0)
        break;
    end

end

% Remove any results below the minimum error
j=find(TempAcc(:,ind_te)>=minerror,1,'first');
if (size(j,1)>0)
    TempAcc=TempAcc(j:1:Rownum-j-1,:);
end
ResultMinErr = TempAcc;

disp('Result for Minimum Error');
disp('r k j q p ADDRows ADDArea ADDErr SUBRows SUBArea SUBErr TOTRows
TOTArea TOTErr');
disp(ResultMinErr);
```

REFERENCES

[1] Roberto Muscedere, “Difficult Operations in the Multi-Dimensional Logarithmic Number System”, PhD Thesis, University of Windsor, 2003.

[2] Steven W. Smith, “The Scientist and Engineer's Guide to Digital Signal Processing”.

[3] Panagiotis D. Vouzis , Sylvain Collange, Mark G. Arnold, “Co-transformation Provides Area and Accuracy Improvement in an HDL Library for LNS Subtraction”, 10thEuromicro Conference on Digital System Design Architectures, Methods and Tools, DSD 2007.

[4] Panagiotis D. Vouzis , Sylvain Collange, Mark G. Arnold, “ LNS Subtraction Using Novel Cotransformation and/or Interpolation”, IEEE International Conference on Application-Specific Systems, Architectures and Processors, ASAP 2007.

[5] Mahzad Azarmehr, “Arithmetic with the Two-Dimensional Logarithmic Number System (2DLNS)”, PhD Thesis, University of Windsor, 2011.

[6] http://en.wikipedia.org/wiki/Logarithmic_number_system

[7] M. Haselman, M. Beauchamp, A. Wood, S. Hauck, K. Underwood, and K. S. Hemmert, “A Comparison of Floating Point and Logarithmic Number Systems for FPGAs”, In Proceedings of the 13th Annual IEEE Symposium on Field Programmable Custom Computing Machines, pages 181–190, Washington, DC, 17–20 April 2005.

[8] Roberto Muscedere, "Improving 2D-log-Number-System Representations by use of an Optimal Base", Eurasip Journal on Advance in Signal Processing, 2008, 1-13, 2008.

[9] J. N. Coleman, E. I. Chester, C. I. Softley and J. Kaldec, "Arithmetic on the European Logarithmic Microprocessor", IEEE Transactions on Computers, vol. 49, no. 7, pp. 702-715, 2000.

[10] N. G. Kingsbury and P. J. Rayner, "Digital Filtering Using Logarithmic Arithmetic", Electronics Letters, vol. 7, pp. 56-58, 1971.

[11] D. M. Lewis, "Interleaved Memory Function Interpolators with Application to an Accurate LNS Arithmetic Unit", IEEE Transactions on Computers, vol. 43, no. 8, pp. 974-982, 1994.

[12] D. M. Lewis, "An Architecture for Addition and Subtraction of Long Word Length Numbers in the Logarithmic Number System", IEEE Transaction on Computers, vol. 39, no. 11, November 1990.

[13] F. J. Taylor, R. Gill, J. Joseph and J. Radke, "A 20 Bit Logarithmic Number System Processor", IEEE Transactions on Computers, vol. 37,pp. 190-200, 1988.

[14] Vassil Dimitrov, Graham Jullien, Roberto Muscedere, "Multiple-Base Number System Theory and Applications", CRC Press 2011.

[15] J. N. Coleman, "Simplification of Table Structure in Logarithmic Arithmetic", IEE Electronic Letters, 31(22):1905–1906, 26 Oct. 1995.

[16] D. M. Lewis, "Interleaved Memory Function Interpolators with Application to and Accurate LNS Arithmetic Unit", IEEE Transactions on Computers, vol. 43, no. 8, pp. 974-982, 1994.

[17] M. G. Arnold, “An Improved Co-transformation for Logarithmic Subtraction”, In Proceedings of the International Symposium on Circuits and Systems, 26–29 May 2002.

[18] R. Muscedere, V. Dimitrov, G.A. Jullien, W.C. Miller, “Efficient Techniques for Binary-to-Multidigit Multidimensional Logarithmic Number System Conversion Using range-Addressable look-Up Tables”, IEEE Transactions on Computers, 54, pp. 257-271, 2005.

[19] R. Muscedere, K. Leboeuf, “A Dynamic Address Decode Circuit for Implementing Range Addressable Look-Up Tables”, IEEE International Symposium on Circuits and Systems, ISCAS 2008.

[20] Mahzad Azarmehr, “A Multi-Dimensional Logarithmic Number System Based Central Processing Unit”, M. A. Sc. Thesis, University of Windsor, 2007.

[21] M. J. Schulte and J. E. Stine, “Symmetric Bipartite Tables for Accurate Function Approximation”, in Proceedings of the 13th IEEE Symposium on Computer Arithmetic, pp. 175–183, Asilomar, CA, July 6–9 1997.

[22] M. G. Arnold, T. A. Bailey, J. R. Cowles, and M. D.Winkel, “Arithmetic Co-transformations in the Real and Complex Logarithmic Number Systems”, IEEE Transactions on Computers, 47(7):777–786, July 1998.

[23] M. G. Arnold., “An Improved Co-transformation for Logarithmic Subtraction”, In Proceedings of the International Symposium on Circuits and Systems (ISCAS’02), pp. 752–755,Scottsdale, Arizona, 26–29 May 2002.

[24] http://flopoco.gforge.inria.fr/

[25] F. de Dinechin and A. Tisserand, “Some Improvements on Multipartite Table Methods. In Proceedings of the 15thSymposium on Computer Arithmetic”, pp. 128–135, Vail, Colorado, 11–13 June 2001.

[26] F. de Dinechin and A. Tisserand, “Multipartite Table Methods”, IEEE Transactions on Computers, 54(3):319–330, March 2005.

VITA AUCTORIS

Leila Sepahi was born in Shiraz, Iran in 1982. She received her Bachelor Degree in Electrical Engineering from Islamic Azad University, Fasa, Iran in 2004. She worked for different engineering companies in Iran for 6 years. In January 2010 she started her Master of Engineering program in University of Windsor. In January 2011 after successfully passing courses needed for M.Eng. she transferred to Master of Applied Science in University of Windsor and started her research under supervision of Dr. R. Muscedere. Her research interests are Computer Arithmetic, VLSI circuit design and Digital Signal Processing.