Improving Accuracy in Logarithmic Multiplication using Operand Decomposition

Mahalingam Venkataraman

University of South Florida

Follow this and additional works at: https://scholarcommons.usf.edu/etd

Part of the American Studies Commons

Scholar Commons Citation

https://scholarcommons.usf.edu/etd/893

This Thesis is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact scholarcommons@usf.edu.
Improving Accuracy in Logarithmic Multiplication using Operand Decomposition

by

Mahalingam Venkataraman

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science and Engineering
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: N. Ranganathan, Ph.D.
Soontae Kim, Ph.D.
Justin E. Harlow III, M.S.

Date of Approval:
March 28, 2005

Keywords: Logarithmic Number Systems, Digital Signal Processing, Mitchell’s Algorithm,
Convolution, Average Error Percentage

© Copyright 2005, Mahalingam Venkataraman
ACKNOWLEDGEMENTS

I would like to thank Dr. N. Ranganathan for giving me this opportunity to work with him. I am grateful to him for his enthusiasm for research, moral support and steadfast friendship. Without his patient guidance this work would not have been possible. He is an ideal role model for aspiring students and I am lucky to have known and worked with him. I also thank Dr. Soontae Kim & Prof. Justin E. Harlow III for the valuable time they took to review this thesis and their helpful comments.
I am very grateful to my loving Father, Mother, and Jayanthi without whose moral support this work would not have reached the completion. I am indebted to them for being a perpetual source of inspiration and motivation for me. Finally, I cannot forget the help and support of all my friends and colleagues who have been with me in every step of this process. Special thanks to Narender Hanchate, Pradeep Fernando and Vasant Manohar, who have always been with me in my hard times.
TABLE OF CONTENTS

LIST OF TABLES iii
LIST OF FIGURES iv
ABSTRACT vi

CHAPTER 1 INTRODUCTION 1
1.1 Digital Signal Processing 1
1.2 LUT Based Logarithmic Interpolation 2
1.3 Mitchell’s Algorithm 3
1.4 Error Correction Approaches to Logarithmic Multiplication 5
   1.4.1 Divided Approximation 5
   1.4.2 Table of Correction Values 6
   1.4.3 Addition of Correction Factor to Final Product 7
1.5 Proposed Operand Decomposition Approach 7
1.6 Thesis Organization 8

CHAPTER 2 RELATED WORK 9
2.1 LUT Based Logarithmic Multiplication 10
2.2 Mitchell’s Multiplication Error 11
2.3 Divided Approximation Based Correction 13
   2.3.1 Hall’s Correction Coefficients 14
   2.3.2 Abed’s Low Overhead Architecture 15
2.4 Correction Term Based Approach 16
   2.4.1 Correction Term Added to Final Product 16
   2.4.2 Table of Correction Values 17

CHAPTER 3 PROPOSED OPERAND DECOMPOSITION APPROACH 18
3.1 Operand Decomposition (OD) 18
   3.1.1 Why Operand Decomposition 20
3.2 Functional Architecture of OD-Mitchell 20
3.3 Combining OD with Divided Approximation 23
3.4 Combining OD with Error Correction Based Approaches 25
   3.4.1 OD with Table of Correction Values (TCV) 25
   3.4.2 Combining OD with Mitchell’s Correction Term 25
3.5 Summary 27
LIST OF TABLES

Table 1.1  LUT Based Interpolation Methods  2
Table 1.2  Mitchell’s Logarithmic Multiplication Algorithm  4
Table 2.1  Table of Correction Values [9]  17
Table 3.1  OD-Mitchell’s Logarithmic Multiplication Algorithm  22
Table 3.2  OD-DA Mitchell’s Logarithmic Multiplication Algorithm  24
Table 3.3  OD-TCV Logarithmic Multiplication Algorithm  26
Table 3.4  Estimated OD-TCV Values  27
Table 4.1  Mitchell’s Vs Operand Decomposed (OD)-Mitchell’s: Error Percentage Level  29
Table 4.2  Mitchell’s Vs Operand Decomposed(OD)-Mitchell’s: Average Error Percent (AEP)  29
Table 4.3  Divided Approximation Vs OD-DA: Error Percentage Level  30
Table 4.4  Divided Approximation Vs OD-DA: Average Error Percent (AEP)  31
Table 4.5  Correction Term Vs OD-Correction Term: Error Percentage Level  33
Table 4.6  Correction Term Vs OD-Correction Term: Average Error Percent (AEP)  33
Table 4.7  Gaussian Smoothing: Error Comparison  33
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Functional Block Diagram of Mitchell Algorithm [17]</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Mitchell’s Logarithm Multiplication Example</td>
<td>4</td>
</tr>
<tr>
<td>1.3</td>
<td>Mitchell’s Logarithmic Approximation</td>
<td>6</td>
</tr>
<tr>
<td>2.1</td>
<td>Taxonomy of Logarithm Based Multiplication</td>
<td>10</td>
</tr>
<tr>
<td>3.1</td>
<td>Operand Decomposition Example</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>Functional Block Diagram of OD-Mitchell’s Architecture</td>
<td>21</td>
</tr>
<tr>
<td>3.3</td>
<td>OD-Mitchell’s Example</td>
<td>22</td>
</tr>
<tr>
<td>3.4</td>
<td>Mitchell’s Example</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>Error Spread of Mitchell’s Algorithm</td>
<td>34</td>
</tr>
<tr>
<td>4.2</td>
<td>Error Spread of Operand Decomposed-Mitchell</td>
<td>34</td>
</tr>
<tr>
<td>4.3</td>
<td>Error Spread of Divided Approximation</td>
<td>34</td>
</tr>
<tr>
<td>4.4</td>
<td>Error Spread of Operand Decomposed - DA</td>
<td>35</td>
</tr>
<tr>
<td>4.5</td>
<td>Error Spread of Table of Correction Values</td>
<td>35</td>
</tr>
<tr>
<td>4.6</td>
<td>Error Spread of Operand Decomposed - TCV</td>
<td>35</td>
</tr>
<tr>
<td>4.7</td>
<td>Error Spread of Mitchell’s Error Correction Term</td>
<td>36</td>
</tr>
<tr>
<td>4.8</td>
<td>Error Spread of Operand Decomposed - MEC</td>
<td>36</td>
</tr>
<tr>
<td>4.9</td>
<td>Discrete Approximation to Gaussian Distribution with standard deviation (σ) = 1.0</td>
<td>36</td>
</tr>
<tr>
<td>4.10</td>
<td>Base Image</td>
<td>37</td>
</tr>
<tr>
<td>4.11</td>
<td>Base Image with Salt and Pepper Noise</td>
<td>38</td>
</tr>
<tr>
<td>4.12</td>
<td>Gaussian Smoothing with Standard Multiplication</td>
<td>38</td>
</tr>
<tr>
<td>4.13</td>
<td>Gaussian Smoothing with Mitchell’s Algorithm</td>
<td>39</td>
</tr>
</tbody>
</table>
IMPROVING ACCURACY IN LOGARITHMIC MULTIPLICATION USING OPERAND DECOMPOSITION

Mahalingam Venkataraman

ABSTRACT

The arithmetic operations such as multiplication and division in binary number system are computationally complex in terms of area, delay and power. Logarithmic Number Systems (LNS) offer a viable alternative combining the simplicity of fixed point number systems and the precision of floating point number systems. However, the computations in LNS result in some loss of accuracy and thus, are limited to mostly signal processing applications; where in certain amount of error is tolerable. In LNS, the cost of computations can be tradeoff with the level of accuracy needed. The Mitchell algorithm proposed in [17], is a simple approach commonly used for logarithmic multiplication. The method involves a high error margin due to a piecewise straight line approximation of the logarithm curve. Thus, several methods have been proposed in the literature for improving the accuracy of Mitchell’s algorithm.

In this thesis, we propose a new method for improving the accuracy of Mitchell’s logarithmic multiplication using operand decomposition. The operand decomposition process decreases the number of bits with the value of ‘1’ in the multiplicands and reduces the amount of approximation. The proposed method brings down the average error percentage of the Mitchell’s logarithmic multiplication by around 45%. It can be combined with previous methods to further improve the accuracy. Experimental results are presented to show that both the error range and the average error percentage can be significantly improved by using operand decomposition.
CHAPTER 1
INTRODUCTION

Several computer systems have been built using logarithmic number systems (LNS), which offers fast and inexpensive multiplication and division regardless of word size. Swartzlander in [24], was the first to develop a four function (addition, subtraction, multiplication and division) arithmetic processor using the logarithmic number system. Several other researchers, [23, 12], have displayed the use of LNS in digital filtering, Fast Fourier Transforms and many other signal processing applications.

Common methods in terms of computer hardware efficiency and power for performing logarithmic multiplication can be classified as look up table based interpolation [4, 7] and Mitchell’s [17] algorithm based method [17]. Earlier studies have shown that interpolation based methods efficiently implement logarithmic operations with a precision and range equivalent to 32 bit IEEE 754 floating point. However, LUT based interpolation requires orders of magnitude more hardware when compared to Mitchell’s logarithm. Mitchell’s [17] logarithmic multiplication algorithm is simple to implement, requiring only shifting and counting operations. The approach however has high error percentage, due to a single straight line approximation in numbers between powers of two. The main focus of this thesis is on improving the accuracy of Mitchell’s [17] logarithmic multiplication algorithm. The context for our work is presented in this chapter.

1.1 Digital Signal Processing

Digital Signal Processing (DSP) is used widely in multimedia systems, telecommunication systems, automobiles and many other mobile processors. Many of these DSP applications like, Finite Impulse Response (FIR) filters, Fast Fourier Transform and speech recognition,
Table 1.1. LUT Based Interpolation Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Author and Year</th>
<th>Word Size (Bits)</th>
<th>ROM Size (Bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM Based Logarithm Multiplication [7]</td>
<td>Brubaker and Becker, 1975</td>
<td>n</td>
<td>$2^n \times 2$</td>
</tr>
<tr>
<td>LNS Processor [23]</td>
<td>Taylor et al., 1988</td>
<td>29</td>
<td>154 K</td>
</tr>
</tbody>
</table>

on the average involve as many multiplication operations as addition operations and sometimes even more [23, 20]. Hence, there is a strong need for high speed, area and power efficient multipliers for digital signal processing.

Another important property of the DSP applications that makes it suitable for LNS processing is its tolerance for error. Often the inputs to these systems are of limited precision and inaccurate due to noise effects. Consequently, an exact result after multiplication is not always required and a rounded product is acceptable in many cases. Hence the area, delay and speed advantage of logarithmic number systems can be traded off against the accuracy required by the DSP system.

1.2 LUT Based Logarithmic Interpolation

Logarithms in mathematics are usually computed using Taylor’s series. Another approach was to store complete table of logarithms for all the numbers. These methods are inefficient in terms of hardware requirement, power and delay overheads. Common approaches in terms of computer hardware efficiency and power to calculating logarithms can be classified as Look up Table (LUT) based interpolation [4, 7] and Mitchell’s algorithm based [17] logarithms. LUT based methods have high accuracy but require more hardware compared to Mitchell’s algorithm based calculation. The hardware overhead in terms of ROM size of various LUT based interpolation methods is given in Table 1.1. On the other hand, Mitchell’s algorithm does not require a ROM and performs multiplication using shifting and counting operations. Hence there is strong interest for Mitchell’s algorithm based calculation among LNS and DSP researchers. Mitchell’s method of approximating logarithms and antilogarithms is briefly explained in the next section.
1.3 Mitchell’s Algorithm

The algorithm for multiplying two numbers using logarithms is straightforward. The logarithms of the input numbers are added and the antilogarithm of the sum is determined. The method used to find the logarithm and the antilogarithm impacts accuracy. Mitchell presented a simple method to approximate the logarithm and antilogarithm [17], using a piecewise straight line approximation of the logarithm curve. Figure 1.1 shows the functional block diagram of the Mitchell’s logarithmic multiplication algorithm.

![Functional Block Diagram of Mitchell Algorithm](image)

Figure 1.1. Functional Block Diagram of Mitchell Algorithm [17]

The Mitchell’s logarithm and antilogarithm calculations required shifting and counting operations only. CMOS VLSI implementation of Mitchell’s algorithm based logarithmic multiplication is reported in [21, 1]. In these approaches, the logarithms of these numbers are calculated separately and then added. The antilogarithm of the summation gives the product of the numbers. The block also requires a zero detector to ensure a zero output if any of the input numbers is zero. Mitchell in his paper [17], proposed the following sequential procedure (Table 1.3) for multiplying two numbers A and B.

The leading one (‘1’) bit position of a number is identified by shifting the numbers left, until the most significant bit is a ‘1’ and decrementing the value of counter, initially loaded with the width (word size) of the input operand, for each shift. The final value of these
Table 1.2. Mitchell’s Logarithmic Multiplication Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Calculate $k_a \leftarrow$ leading one position of number $A$;</td>
</tr>
<tr>
<td>2</td>
<td>Calculate $k_b \leftarrow$ leading one position of number $B$;</td>
</tr>
<tr>
<td>3</td>
<td>Calculate $m_a \leftarrow$ remaining part of the operand $A$;</td>
</tr>
<tr>
<td>4</td>
<td>Calculate $m_b \leftarrow$ remaining part of the operand $A$;</td>
</tr>
<tr>
<td>5</td>
<td>Append as floating point numbers $f_a \leftarrow (k_a, m_a)$ and $f_b \leftarrow (k_b, m_b)$;</td>
</tr>
<tr>
<td>6</td>
<td>Add the logarithm values and then calculate the antilogarithm;</td>
</tr>
<tr>
<td>7</td>
<td>Calculate the product $A \times B \leftarrow \text{antilog}(f_{ab})$;</td>
</tr>
</tbody>
</table>

Counters are stored in registers $k_a$ and $k_b$. Append the remaining part of the input numbers, $m_a$ and $m_b$ with $k_a$ and $k_b$ as the mantissa part for the logarithms, $f_a$ and $f_a$. Add these logarithm values and decode the characteristic value of the result to insert an one (‘1’) into appropriate position of the result. The remaining part of the added result $f_{ab}$ is then appended immediately after this one (‘1’). An example for illustrating the above procedure is shown in figure 1.2. Figure shows the logarithmic multiplication of decimal numbers 10 and 15.

18 = 00010010  
60 = 00111100  
log 18 = 100.0010  
log 60 = 101.1110  
log 18 + log 60 = 1010.0000  
Result = 1000000000  
Decimal = 1024

Figure 1.2. Mitchell’s Logarithm Multiplication Example

It can be seen from the figure that, the location of the most significant one (‘1’) bit determines the integer part and the remaining bits become the fraction part for $\log_2{18}$ and $\log_2{60}$. The two numbers $\log_2{18}$ and $\log_2{60}$ are then added. A one (‘1’) is placed on the final product by decoding the integer value of the summation and the remaining fraction part is appended immediately after this one (‘1’). The above discussion shows that
approximate logarithm multiplication is simple to perform and requires only shifting and counting operations. But the example shown has an error of 5%. The maximum possible error for Mitchell's logarithm multiplication is around 11.1% and average error is around 3.8%. Many researchers in the past, have attempted to improve the precision of Mitchell's algorithm [17] based logarithms. In the next section, we discuss some works that attempt to reduce the error in Mitchell's algorithm.

1.4 Error Correction Approaches to Logarithmic Multiplication

Mitchell's algorithm approximated the logarithm curve by piecewise linear approximations in intervals between powers of 2 integers. In other words the true logarithm and Mitchell's logarithm intersect whenever the logarithm value is an integer. Due to its low area, power and delay overhead, several researchers have tried to improve the accuracy of Mitchell's algorithm based logarithmic multiplication. The correction methods proposed in the literature can be broadly classified as,

1. Divided straight line approximation for logarithm and antilogarithm calculation.
2. Table of correction values to be added to the logarithm summation.
3. Analytical correction term to be added to the final product.

1.4.1 Divided Approximation

The piecewise straight line approximation of logarithms by Mitchell's algorithm is shown in Figure 1.3. Here truelog refers to the actual logarithm and Mitchell's approximation is shown as mitchell log. The error in the Mitchell's logarithm is due to the fractional part of the logarithm. Hence, whenever the fraction part is zero the log mitchell curve intersects with truelog. The logarithm error due to Mitchell's approximation is in the range 0 \leq Error \leq 0.08639 and attains the maximum value when fraction part of the logarithm is equal to 0.44.

The piecewise straight line approximation shown in figure 1.3 can be analytically represented as,
\[ a \times x + b \]  \hspace{1cm} (1.1)

where \( a = 1 \) and \( b = 0 \) and 'x' refers to the value of the fraction part of the logarithm. Mitchell’s method had a large error because of a single straight line approximation. Other researchers in [2, 10, 8, 22], divided the interval between these two integers in regions of 2, 4 and 8 to improve the accuracy. A separate logarithmic approximation in each of these regions depending on the value of the fraction part is proposed. The accuracy of these methods is proportional to the design complexity. Similar divided approximation strategy has also been proposed for antilogarithm calculation by Abed, in [3].

1.4.2 Table of Correction Values

Mclaren in [9], proposed a simulation based correction strategy to Mitchell’s algorithm. The method was based on the same concept of divided approximation, but the author in, [9], provided a constant table of correction values for different fractional regions. This correction term is added to the logarithm summation and is determined by the value of fraction part of the logarithm numbers. The fraction part of the logarithms is divided into
eight regions in steps of 0.125 and 64 correction values are stored in the table. The method reduced the average error percentage of Mitchell’s logarithms to around 0.03%  [9].

1.4.3 Addition of Correction Factor to Final Product

Mitchell in his paper  [17], analyzed the error due to the logarithm and antilogarithm approximations and developed an analytical correction expression. The correction expression to be used is dependent on whether there is a carry over from the fraction to the integer part in the logarithm summation. The correction expression required two extra additions only. The method reduced the maximum error of Mitchell’s logarithm multiplication to 2.8%.

In this thesis, we present a novel approach to improve the accuracy of logarithmic multiplication. The proposed method improves accuracy by itself and also operates well when coupled with the above listed error correction approaches.

1.5 Proposed Operand Decomposition Approach

In this work, operand decomposition is used for improving the accuracy in logarithmic multiplication. A similar decomposition of input operands has been reported by the authors in  [11], to reduce the switching activity of standard multiplication. The decomposition breaks the input operands X and Y into four numbers A, B, C and D. The product of X and Y is then calculated using the following formula.

\[ X \times Y = ((A \times B) + (C \times D)) \]

(1.2)

The decomposed operands have less number of one (‘1’) bits than the original input. In fact the probability of a bit being one (‘1’) in the decomposed operands is (1/4) compared to a (1/2) binary inputs X and Y. The method reduces the switching activity of array multipliers by around 12%.
The operand decomposition was taken up for improving logarithmic multiplication’s accuracy for the following reasons.

1. Mitchell proved that the error in Mitchell’s logarithm is due to the fractional part. Intuitively if we increase the number of zero (‘0’) bits in the binary number, this will decrease the value of the fraction part and also the error.

2. Divided approximation methods, use different coefficients to closely follow the logarithmic curve. Operand decomposition can be combined with divided approximation to give more flexibility in selecting optimal coefficients for maximum error reduction.

3. Separate table of correction values can be used for the decomposed multiplications to follow the curve more closely.

In accordance with our assumption the operand decomposition approach reduced the error in logarithm multiplication. Efficient algorithms for combining operand decomposition with the previous error correction approaches were developed. One such combination of the proposed approach with a newly generated table of correction values reduces the average error percentage to 0.005%. Secondly, the percentage of multiplications with errors below 1 and 2 percentages increased for all the methods.

1.6 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 explains the various correction strategies proposed for Mitchell’s logarithm and antilogarithm calculation. The improvement to the error percentage of logarithmic multiplication when using operand decomposition strategy with and without average slack error based correction methods is presented in Chapter 3. In Chapter 4, we present the simulation methodology and the experimental results for the proposed algorithms. We conclude the thesis in Chapter 5.
CHAPTER 2

RELATED WORK

Many digital signal processing and control systems involve heavy use of arithmetic operations like addition, subtraction, multiplication and division. The use of fixed point number system for these operations is fast and precise, but not power efficient for multiply intensive applications [20]. Fixed point numbers also have the problem of overflow and scaling. This is a serious concern in mobile and multimedia applications [6]. Floating point number systems maintain a relative precision and overcome the overflow and scaling problems. However, floating point number systems require complex hardware and consume more power than fixed point number systems.

Logarithmic number system (LNS) offers the advantages of both fixed point and floating point number systems. Swartzlander and Kingsbury [23, 12], were the first to develop the use of LNS in digital filtering and FFT applications respectively. LNS are particularly suited to cases, where precision requirements are less and multiplication is more frequent than addition. LNS based multiplication methods can be broadly classified as look up table based and Mitchell’s algorithm based multiplication. The latter method has very small overhead but has a high error percentage. Several improvements have been proposed to Mitchell’s algorithm based multiplication. A taxonomy diagram showing, selected research done in this context is given in figure 2.1.

Figure (2.1) classifies logarithmic multiplication to table based logarithms and Mitchell’s based logarithms. The low overhead of Mitchell’s algorithm makes it viable for digital signal processing. Since the main focus of this thesis is on Mitchell’s algorithm, the look up table based approaches is only visited superficially in this chapter. Secondly, the correction methods to Mitchell’s algorithm namely, divided approximation and table of correction
value methods are explained in detail. The characteristics of these methods were studied carefully and combined efficiently with our operand decomposition strategy.

2.1 LUT Based Logarithmic Multiplication

An early approach to logarithmic multiplication utilized complete logarithm value tables stored in memory. This was however, inefficient in terms of power and hardware complexity. Look up Table (LUT) based interpolation techniques proved to be an efficient strategy to reduce LUT size at the cost of accuracy. The pioneering work in this area of LUT based interpolation, was described by E.E. Swartzlander, in [24]. The authors in [24] presented algorithms for performing four basic arithmetic operations (addition, subtraction, multiplication and division) with the LNS. Several LUT based interpolation techniques [7, 13, 16, 20] followed this, by identifying optimized LUT sizes for a specified accuracy.

More recently a new form of logarithmic arithmetic termed as the multidimensional logarithmic number system (MDLNS) [19, 18] became popular. The MDLNS provides
more degrees of freedom than the LNS by the use of orthogonal bases and the ability to gain from the use of multiple digits. But these advantages increase the complexity of the conversion process. Another solution [4, 5] in this area targeted redundancy to reduce LUT size.

Even after several optimizations, the sizes of logarithm tables are still very high. Hence these methods are not preferred in applications where low overhead is very important and accuracy to an extent is tolerable. Mitchell’s algorithm is useful in such cases. Numerous works have been attempted so far to increase the accuracy and usage of Mitchell’s logarithms in signal processing contexts. This thesis is yet another step towards that objective. Mitchell showed in his paper [17], that the error in his approach is due to the fraction component of the two multiplicands. The following section repeats this logarithm error percentage analysis due to Mitchell’s method.

2.2 Mitchell’s Multiplication Error

Consider a binary number N in the range $2^{k+1} > N \geq 2^j$ and $k \geq j$. The Number N can be written as,

$$N = \sum_{i=j}^{k} 2^i Z_i$$  \hspace{1cm} (2.1)

Since $Z_k$ is the most significant bit, we may assume $Z_k = 1$ for any valid $k \geq j$, without loss of generality. Factoring out the value $2^k$ from N, we get

$$N = 2^k (1 + \sum_{i=j}^{k-1} 2^{i-k} Z_i)$$  \hspace{1cm} (2.2)

Let the second term in the above equation be referred to x, as given below.

$$\sum_{i=j}^{k-1} 2^{i-k} Z_i = x$$  \hspace{1cm} (2.3)
Since \( k \geq j \), the value of \( x \) will be in the range \( 0 \leq x < 1 \) and will be referred to as the fraction term of the binary number \( N \) in the rest of the thesis. The number \( N \) can now be represented as,

\[
N = 2^k (1 + x)
\]  
(2.4)

The true logarithm of this binary number and Mitchell’s approximation can be given as,

\[
\log_2(N)_{\text{true}} = k + \log_2(1 + x)
\]  
(2.5)

\[
\log_2(N)_{\text{approx}} = k + x
\]  
(2.6)

It can be seen from the above equation, that Mitchell’s method approximates the \( \log(1+x) \) value with the value of the fraction \( x \), hence eliminating the need for look up tables for multiplying numbers. This straight line approximation of logarithms over power of two intervals is simple and requires only shifting and counting operations. The true and Mitchell’s logarithm of the products can be given as the sum of the individual logarithms,

\[
\log_2(N_1 \cdot N_2)_{\text{true}} = k_1 + \log_2(1 + x_1) + k_2 + \log_2(1 + x_2)
\]  
(2.7)

\[
\log_2(N_1 \cdot N_2)_{\text{approx}} = k_1 + x_1 + k_2 + x_2
\]  
(2.8)

The Mitchell’s logarithmic product equation can be divided into two separate equations based on whether there is a carry from the summation of the logarithms. The new equations for the logarithm of the product term is given by,

\[
\log_2(N_1 \cdot N_2)_{\text{approx}} = k_1 + k_2 + x_1 + x_2, \text{ when } x_1 + x_2 < 1
\]  
(2.9)

\[
\log_2(N_1 \cdot N_2)_{\text{approx}} = 1 + k_1 + k_2 + (x_1 + x_2 - 1), \text{ when } x_1 + x_2 \geq 1
\]  
(2.10)
The antilogarithmic approximation of the above summation gives the final product, which is given by,

\[ P_{\text{approx}} = 2^{k_1+k_2}(1 + x_1 + x_2), \text{ when } x_1 + x_2 < 1 \]  
\[ P_{\text{approx}} = 2^{1+k_1+k_2}(x_1 + x_2), \text{ when } x_1 + x_2 \geq 1 \]  

(2.11)  
(2.12)

The error \( E_m \) in the Mitchell’s logarithm multiplication is then defined as,

\[ E_m = \frac{P_{\text{approx}} - P_{\text{true}}}{P_{\text{true}}} \]  

(2.13)

Where, \( P_{\text{true}} \) is defined as the actual product calculated using efficient table look up method or using a standard array multiplier. The maximum possible multiplication error using Mitchell’s method is around -11.1 percent and will occur when both the fraction parts are equal to 0.45. The error is always negative for multiplication. Hence, they might get compounded if successive multiplication is performed on the data. Several researches in the literature have tried to reduce this error percentage either by using separate approximation in different ranges or adding an offset value to Mitchell’s logarithmic multiplication. Some of the important works are briefly reviewed here.

2.3 Divided Approximation Based Correction

Mitchell in his paper, approximated the logarithm curve by a piecewise linear curve in power of two intervals. The authors in [10, 2, 3] divided these intervals into various regions and used different equations to improve the accuracy of logarithmic multiplication. In this section we describe Hall et al’s. [10] optimal equations to Mitchell’s approximation and abed et al’s. [2, 3] low overhead correction equations.
2.3.1 Hall’s Correction Coefficients

Hall et al. in [10], proposed a divided approximation method for improving the accuracy of Mitchell’s algorithm based logarithmic multiplication. Mitchell’s algorithm approximated the logarithm curve by piecewise linear approximation in intervals between powers of 2 integers as shown in Figure 1.3. Hall [10] and Combet [8] partitioned the range of fraction \((x)\) into four parts and making a separate approximation in each range of the fraction. The equations developed in combet were reportedly found using trial and error for easy hardware implementation. Hall, on the other hand selected optimal coefficients to minimize mean square error rather than selecting by trial and error. Coefficients are carefully chosen to be fractions with integer numerators and powers of two denominators. Hall’s four interval equations for logarithm approximation is given below,

\[
\log_2(1 + x)' = x + \frac{37}{128} x + \frac{1}{128} \quad \text{for} \; x \in [0.00, 0.25]
\]  

\[
\log_2(1 + x)' = x + \frac{3}{64} x + \frac{1}{16} \quad \text{for} \; x \in [0.25, 0.50]
\]  

\[
\log_2(1 + x)' = x + \frac{7}{64} x' + \frac{1}{32} \quad \text{for} \; x \in [0.50, 0.75]
\]  

\[
\log_2(1 + x)' = x + \frac{29}{128} x' \quad \text{for} \; x \in [0.75, 1.00]
\]

where \(x' = (1 - x)\).

Similarly Hall et al. [10] also proposed four region correction equations for antilogarithm calculation. Once again mean square error is used as the error minimization criteria. The four equations for Mitchell’s antilogarithm approximation by hall is given below,

\[
\log_2(1 + x)' = x + \frac{1}{4} x' + \frac{3}{4} \quad \text{for} \; x \in [0.00, 0.25]
\]  

\[
\log_2(1 + x)' = x + \frac{13}{128} x' + \frac{55}{64} \quad \text{for} \; x \in [0.25, 0.50]
\]  

\[
\log_2(1 + x)' = x + \frac{9}{128} x + \frac{7}{8} \quad \text{for} \; x \in [0.50, 0.75]
\]  

14
\[
\log_2(1 + x)' = x + \frac{35}{128} x + \frac{23}{32} \quad \text{for } x \in [0.75, 1.00] \quad (2.21)
\]

where \(x' = (1 - x)\).

Hall’s method is more accurate and reduces the error percentage of logarithmic multiplication to one fourth of the Mitchell’s method. However, hall’s method has a large hardware overhead and is difficult to implement. In both combet’s and hall’s method [10, 8] all the bits of the fraction part are used in the correction term. Abed et al. [2] presented a low overhead correction approach for Mitchell’s logarithm multiplication. The approach uses only selected bits of the mantissa for correction and only, power of two coefficients for both numerator and denominator. Such coefficients only require shifting operations for scaling. Abed’s approach is explained in the next section.

### 2.3.2 Abed’s Low Overhead Architecture

Abed et al. [2, 3] developed a binary logarithm correcting algorithm, that can be implemented with small and fast circuitry and also have a high accuracy. Three different correction strategies namely, 2-region, 3-region and 6-region equations are proposed with varying hardware complexity trading off maximum error percentage. They used only selected bits of the mantissa and powers of two values for coefficients of the linear equations. As an example illustration, the equation for the 2-region correcting algorithm is given below. Here, Mitchell’s straight line approximation is divided into two fractional regions and piecewise linearly approximated with the following equations.

\[
\log_2(1 + x)' = x + \frac{1}{4} x_{3\text{MSBits}} \quad \text{for } x \in [0.0, 0.5] \quad (2.22)
\]

\[
\log_2(1 + x)' = x + \frac{1}{4} x_{3\text{MSBits}}' \quad \text{for } x \in [0.5, 1.0] \quad (2.23)
\]

where \(x' = (1 - x_{3\text{MSBits}} - 2^{-3})\).

An important point to note is that only 3 Most Significant Bits (MSB) of the mantissa is used for correction and the coefficients of the equations are power of two. Hence only
a simple shifting and adding is required for generating the correction terms. The paper also presents a single cycle combinational logic implementation for the proposed correction algorithm. The 2-region, 3-region and 7-region correction approach reduces the error percentage of binary to logarithm conversion error to 1.48, 0.699 and 0.306 percentages respectively. Similar to hall’s [10] approach abed et al. [3] also proposed correction equations for antilogarithm conversion. Once again the authors here use better coefficients and only selected mantissa bits for efficient accuracy improvement.

A second class of accuracy improvement in logarithmic multiplication is classified as correction term based approach. In the next section, we discuss two important methods namely, addition of Mitchell’s correction term and table of correction values by Mitchell and McLaren respectively, in [17, 9].

2.4 Correction Term Based Approach

In this section, we briefly discuss about the correction term based approaches proposed for improving accuracy of Mitchell’s multiplication algorithm. The methods are based on adding a correction term either to the final product as in [17] or to the logarithm summation as in [9].

2.4.1 Correction Term Added to Final Product

Mitchell in [17], proposed a correction term to reduce the error percentage in his algorithm. The author [17] developed two analytical models for correction terms to be added to the final product. The carry bit from the logarithmic addition of two multiplicands decides the usage of appropriate correction term. The new equations for Mitchell’s logarithmic multiplications are shown below.

\[ P_{\text{approx}} = 2^{k_1}x_1(1 + x_1 + x_2) + 2^{k_1+k_2}x_1x_2 \text{ when } x_1 + x_2 < 1 \]  \hspace{1cm} (2.24)

\[ P_{\text{approx}} = 2^{1+k_1+k_2}(x_1 + x_2) + 2^{k_1+k_2}y_1y_2 \text{ when } x_1 + x_2 \geq 1 \]  \hspace{1cm} (2.25)
where, \( y_1 = 1 - x_1 \) and \( y_2 = 1 - x_2 \).

The new equations with the correction terms can be easily generated with just two extra additions. Identify, whether the carry bit is zero (‘0’) or one (‘1’). Suitably generate either \( x_1x_2 \) or \( y_1y_2 \) using an addition operation. Scale the correction term by the factor \( 2^{k_1+k_2} \) and add it to the original result. The correction term requiring two additions (one to perform the multiplication and the other to add to original result) reduced the maximum error percentage of Mitchell’s multiplication to 2.8 percentage.

### 2.4.2 Table of Correction Values

Mclaren in \([9]\), devised a simple method of adding a table of constant values to the logarithm summation. The author divided the fraction part of the logarithm inputs into 8 ranges in steps of 0.125 and a correction term was stored in a table for each combination. The constant table of correction values is found out by averaging out error values after extensive simulation. The table of correction values used in \([9]\) is shown in Table 2.1.

Average percentage error is used as the optimization metric in this simulation based approach. The table of correction values approach reduced the average percent error of Mitchell’s logarithm from 3.8% to 0.03%. In this research work, we devise a new operand decomposition strategy to improve the accuracy of Mitchell’s logarithmic multiplication. The proposed operand decomposition approach is given in the next chapter.

<table>
<thead>
<tr>
<th>0.000</th>
<th>0.0125</th>
<th>0.125</th>
<th>0.250</th>
<th>0.375</th>
<th>0.500</th>
<th>0.625</th>
<th>0.750</th>
<th>0.875</th>
<th>1.000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>0.0048</td>
<td>0.0131</td>
<td>0.0198</td>
<td>0.0246</td>
<td>0.0293</td>
<td>0.0331</td>
<td>0.0368</td>
<td>0.0405</td>
<td>0.0442</td>
</tr>
<tr>
<td>0.125</td>
<td>0.0131</td>
<td>0.0260</td>
<td>0.0364</td>
<td>0.0468</td>
<td>0.0572</td>
<td>0.0676</td>
<td>0.0780</td>
<td>0.0884</td>
<td>0.0988</td>
</tr>
<tr>
<td>0.250</td>
<td>0.0198</td>
<td>0.0548</td>
<td>0.0833</td>
<td>0.1118</td>
<td>0.1403</td>
<td>0.1688</td>
<td>0.1973</td>
<td>0.2258</td>
<td>0.2543</td>
</tr>
<tr>
<td>0.375</td>
<td>0.0246</td>
<td>0.0706</td>
<td>0.1287</td>
<td>0.1868</td>
<td>0.2449</td>
<td>0.3030</td>
<td>0.3611</td>
<td>0.4192</td>
<td>0.4773</td>
</tr>
<tr>
<td>0.500</td>
<td>0.0331</td>
<td>0.0833</td>
<td>0.1327</td>
<td>0.1821</td>
<td>0.2315</td>
<td>0.2809</td>
<td>0.3303</td>
<td>0.3807</td>
<td>0.4311</td>
</tr>
<tr>
<td>0.625</td>
<td>0.0405</td>
<td>0.1000</td>
<td>0.1600</td>
<td>0.2199</td>
<td>0.2798</td>
<td>0.3397</td>
<td>0.3996</td>
<td>0.4595</td>
<td>0.5194</td>
</tr>
<tr>
<td>0.750</td>
<td>0.0488</td>
<td>0.1111</td>
<td>0.1714</td>
<td>0.2318</td>
<td>0.2921</td>
<td>0.3525</td>
<td>0.4128</td>
<td>0.4732</td>
<td>0.5335</td>
</tr>
<tr>
<td>0.875</td>
<td>0.0572</td>
<td>0.1222</td>
<td>0.1828</td>
<td>0.2436</td>
<td>0.3044</td>
<td>0.3652</td>
<td>0.4260</td>
<td>0.4868</td>
<td>0.5476</td>
</tr>
<tr>
<td>1.000</td>
<td>0.0656</td>
<td>0.1333</td>
<td>0.1944</td>
<td>0.2556</td>
<td>0.3170</td>
<td>0.3784</td>
<td>0.4400</td>
<td>0.5016</td>
<td>0.5632</td>
</tr>
</tbody>
</table>
CHAPTER 3
PROPOSED OPERAND DECOMPOSITION APPROACH

Mitchell’s logarithmic multiplication algorithm has always been of interest to researchers in digital signal processing for its low hardware complexity. However, the main problem with this algorithm is the high error percentage associated with its piecewise straight line approximation. All the approaches presented in the previous chapter improve the accuracy of Mitchell’s logarithmic multiplication by identifying the average error slack over a certain range. In this research work, we propose a novel operand decomposition approach to improve accuracy of Mitchell’s logarithm multiplication.

In this chapter, we first sketch the fundamentals of operand decomposition approach and then explain why it improves the accuracy of logarithmic multiplication. In the next section, we present the functional architecture of the proposed approach. We then conclude this chapter by proposing algorithms to combine operand decomposition with the error correction approaches presented in the previous chapter.

3.1 Operand Decomposition (OD)

In this section, we explain the operand decomposition approach for multiplying two binary numbers. A similar strategy was used by the authors, in [11], to reduce the switching activity in binary multipliers. Consider two n-bit binary numbers X and Y of the form,

\[ X = [x_{n-1}x_{n-2} \ldots \ldots x_2x_1x_0] \text{, and} \]

\[ Y = [y_{n-1}y_{n-2} \ldots \ldots y_2y_1y_0]. \]

These operands X and Y are decomposed into the following four operands A, B, C and D.
\[ A = [a_{n-1}a_{n-2} \ldots \ldots a_2a_1a_0], \quad (3.3) \]
\[ B = [b_{n-1}b_{n-2} \ldots \ldots b_2b_1b_0], \quad (3.4) \]
\[ C = [c_{n-1}c_{n-2} \ldots \ldots c_2c_1c_0], \text{ and} \quad (3.5) \]
\[ D = [d_{n-1}d_{n-2} \ldots \ldots d_2d_1d_0] \quad (3.6) \]

where, the individual bits in the decomposed operands are calculated using the following equations, \( a_i = x_i \lor y_i, \ b_i = x_i \land y_i, \ c_i = \bar{x}_i \land y_i \) and \( d_i = x_i \land \bar{y}_i \), the product is then computed from the decomposed operands using the following property.

\[ X \ast Y = (C \ast D) + (A \ast B) \quad (3.7) \]

The equation can be easily verified for correctness, using simple substitution and a simple proof is given in [11]. The approach increases the number of zero bits in the decomposed multiplications and hence decreases the switching power in the multiplication operation.

An example illustrating the operand decomposition approach for binary multiplication is given in figure refodexample. The example illustrated, shows the logarithmic multiplication of integers 18 and 60.

\begin{align*}
X &= b'(00010010) = d'18 \\
Y &= b'(00111100) = d'60 \\
A &= b'(00111110) = d'62 \\
B &= b'(00010000) = d'16 \\
C &= b'(00101100) = d'44 \\
D &= b'(00000010) = d'02 \\
X \ast Y &= ((A \ast B) + (C \ast D)) \\
X \ast Y &= ((62 \ast 16) + (44 \ast 2)) \\
X \ast Y &= 1080
\end{align*}

Figure 3.1. Operand Decomposition Example
In this paper we use this simple operand decomposition for improving the accuracy in Mitchell’s logarithmic multiplication.

3.1.1 Why Operand Decomposition

Operand Decomposition approach was investigated with Mitchell’s logarithmic multiplication for the following reasons. It has been proved, in [17, 9], that error in Mitchell’s logarithm calculation is due to the fraction part of the logarithms. Since, the operand decomposition strategy increases the number of zeroes in the multiplicand, the value of the fraction part in logarithm decreases and hence the error.

Secondly, the divided approximation and other correction term based approaches tries to follow the logarithm curve closely by adding the average error slack. With operand decomposition, the flexibility in selecting the coefficients for these equations increases and the decomposed multiplications can coordinate well to follow the curve more closely. In the next section, we explain the functional architecture of the Operand Decomposed (OD)-Mitchell’s logarithm multiplication algorithm.

3.2 Functional Architecture of OD-Mitchell

In this section, we explain the functional level architecture of the proposed Operand Decomposed (OD)-Mitchell’s algorithm. The block diagram of the OD-Mitchell’s architecture is shown in figure 3.2. It consists of six major blocks. They are the decomposition unit, logarithm unit, antilogarithm unit, two binary adders and a zero detector unit.

The internal architectural details of decomposition, logarithm and antilogarithm units are briefly explained here. The decomposition unit is simple to design and only requires $6^n$ 2-input (and, or, not) gates, where $n$ is the number of input bits in the multiplication. The logarithm and the antilogarithm units are completely designed with combinational logic blocks only [2, 3]. The logarithm unit, internally is provided with a leading one detector circuit, an $(n \times \log_2 n)$ ROM and a logarithmic shifter. The antilogarithm unit requires only a single logarithmic shifter. Gate level implementation details of these internal units are
explained in detail by the authors in, [2, 3]. The Operand Decomposed (OD)-Mitchell’s logarithmic multiplication algorithm is shown in Table 3.2. Here the operands X and Y are decomposed to A, B, C and D using the procedure given in section 3.1. The decomposed operands are multiplied separately and added together to obtain the final product.

Figure 3.3 shows an example illustration of the operand decomposed logarithmic multiplication. As shown in the figure, the multiplicands X = d’18 and Y = d’60 are decomposed into four numbers A= d’62, B=d’16, C=d’44 and D=d’02. The decomposed operands are then multiplied separately using Mitchells logarithm multiplication. The results of these decomposed products sum1=d’992 and sum2=d’88 are then added to get the final product X *Y = d’1080. The same logarithmic multiplication example is shown in figure 3.4 without operand decomposition. The final product in this case equal to d’1024 with an error of 5.2%
Table 3.1. OD-Mitchell’s Logarithmic Multiplication Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Decompose the numbers X and Y into A, B, C and D; Percentage ( k_a ) and ( k_b ) are the characteristic of the logarithms of A and B;</td>
</tr>
<tr>
<td>2</td>
<td>Calculate ( k_a ) ( \rightarrow ) leading one position of number A;</td>
</tr>
<tr>
<td>3</td>
<td>Calculate ( k_b ) ( \rightarrow ) leading one position of number B;</td>
</tr>
<tr>
<td>4</td>
<td>Calculate ( m_a ) ( \rightarrow ) remaining part of the operand A;</td>
</tr>
<tr>
<td>5</td>
<td>Calculate ( m_b ) ( \rightarrow ) remaining part of the operand A;</td>
</tr>
<tr>
<td>6</td>
<td>Append as floating point numbers ( f_a ) ( \rightarrow ) (( k_a ), ( m_a )) and ( f_b ) ( \rightarrow ) (( k_b ), ( m_b ));</td>
</tr>
<tr>
<td>7</td>
<td>Add the logarithm values and then calculate the antilogarithm;</td>
</tr>
<tr>
<td>8</td>
<td>Calculate ( P_{ab} ) ( \leftarrow ) antilog(( f_{ab} ));</td>
</tr>
<tr>
<td>9</td>
<td>Perform steps 2 and 3 for decomposed multiplicands C and D and store the values in ( k_c ) and ( k_d );</td>
</tr>
<tr>
<td>10</td>
<td>Calculate ( m_c ) and ( (m_d) ) as in steps 4 and 5;</td>
</tr>
<tr>
<td>11</td>
<td>Append as floating point numbers ( f_c ) ( \rightarrow ) (( k_c ), ( m_c )) and ( f_d ) ( \rightarrow ) (( k_d ), ( m_d ));</td>
</tr>
<tr>
<td>12</td>
<td>( f_{cd} ) ( \leftarrow ) ( f_c ) + ( f_d );</td>
</tr>
<tr>
<td>13</td>
<td>Calculate ( P_{cd} ) ( \leftarrow ) antilog(( f_{cd} ));</td>
</tr>
<tr>
<td>14</td>
<td>Adding the result of the decomposed multiplications;</td>
</tr>
<tr>
<td>15</td>
<td>Product: ( X \times Y ) ( \leftarrow ) ( P_{ab} ) + ( P_{cd} );</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
X &= b'(00010010) = d'18 \\
Y &= b'(00111100) = d'60 \\
A &= b'(00111110) = d'62 \\
B &= b'(00010000) = d'16 \\
C &= b'(00101100) = d'44 \\
D &= b'(00000010) = d'02 \\
\text{lg } A &= 0101.111100 \\
\text{lg } B &= 0100.000000 \\
\text{Sum1} &= \text{lg } A + \text{lg } B = 1001.111100 \\
\text{Antilog(Sum1)} &= 0000001111100000 = d'992 \\
\text{lg } C &= 0101.011000 \\
\text{lg } D &= 0001.000000 \\
\text{Sum2} &= \text{lg } C + \text{lg } D = 0110.011000 \\
\text{Antilog(Sum2)} &= 0000000001011000 = d'88 \\
\end{align*}
\]

Product \( (X, Y) = X \times Y = \text{Antilog(Sum1)} + \text{Antilog(Sum2)} = 1080 \)

Figure 3.3. OD-Mitchell’s Example

approximately. Hence the operand decomposition approach clearly improves the accuracy of logarithmic multiplication. The average error percent of Mitchells logarithmic multiplication, after operand decomposition reduced to 2.1% an improvement of around 45%. Further, in the next section, we propose algorithms to efficiently combine operand decom-
X = b’{00010010} = d’18  
Y = b’{00111100} = b’60  

\[ \log X = 0100.001000 \]
\[ \log Y = 0101.111000 \]
\[ \text{Sum} = \log x + \log Y = 1010.000000 \]
\[ X \times Y = \text{Antilog} (\text{Sum}) = 0000001000000000 = d’1024 \]

Figure 3.4. Mitchell’s Example

position with other error correction approaches presented in the previous chapter to further decrease the error percentage of Mitchell’s algorithm based logarithmic multiplication.

3.3 Combining OD with Divided Approximation

In this section, we propose a new algorithm to combine operand decomposition with Abed’s [2] divided approximation approach for logarithmic multiplication. The divided approximation now has more flexibility as the two separate multiplications can be improved with different correction equations for their mantissa part. Here the operands X and Y are decomposed to A, B, C and D using the procedure given in section 3.1. The decomposed operands A, B and C, D are then multiplied separately using the Mitchell’s logarithmic multiplication procedure. The products of these multiplications \( P_{ab} \) and \( P_{cd} \) are then added to get the final product \( X \times Y \). The algorithm was initially assumed to use two separate equations to approximate the mantissa part in logarithm calculation for OD-DA. This is because, the divided approximation procedure sometimes overestimate the mantissa value of the logarithms and the error increases in the opposite side (high negative value). However, simulations indicate that the same equations for both the decomposed multiplications provide better error range. Hence in this research, we use the following 2-region functions for the decomposed multiplications. The coefficients for these 2-region functions are identified from extensive trial and error simulations. These simulations takes the values of product \( P_{ab} \) and product \( P_{cd} \) into account with average percent error and error range as the im-
Table 3.2. OD-DA Mitchell’s Logarithmic Multiplication Algorithm

Step 1: Decompose the numbers X and Y into A, B, C and D;

$k_a$ and $k_b$ are the characteristic of the logarithms of A and B;

Step 2: Calculate $k_a$ ← leading one position of number A;

Step 3: Calculate $k_b$ ← leading one position of number B;

$m_a$ and $m_c$ refers to the mantissa part of the logarithm of A and B;

Step 4: Calculate $m_a$ ← 2-region improve($m_a$);

Step 5: Calculate $m_b$ ← 2-region improve($m_b$);

Step 6: Append as floating point numbers $f_a$ ← ($k_a.m_a$) and $f_b$ ← ($k_b.m_b$);

% add the logarithm values and then calculate the antilogarithm;

Step 7: $f_{ab} ← f_a + f_b$;

Step 8: Calculate $P_{ab} ← \text{antilog}(f_{ab})$;

Step 9: Perform steps 2 and 3 for decomposed multiplicands C and D and store

the values in $k_c$ and $k_d$;

Step 10: Calculate $m_c$ ← 2-region improve($m_c$);

Step 11: Calculate $m_d$ ← 2-region improve($m_d$);

Step 12: Append $f_c$ ← ($k_c.m_c$) and $f_d$ ← ($k_d.m_d$) as a number;

Step 13: $f_{cd} ← f_c + f_d$;

Step 14: Calculate $P_{cd} ← \text{antilog}(f_{cd})$;

% Adding the result of the decomposed multiplications;

Step 15: Product: $X \times Y ← P_{ab} + P_{cd}$;

Step 16: function 2-region improve($m$);

% 2 region approximation for log($m$) of operands A and B;

Step 17: if (m < 0.5)

Step 18: $m ← m + (\frac{1}{2})m$

Step 19: else

Step 20: $m ← m + (\frac{1}{2})(1 - m - \frac{1}{2})$

Step 21: return m;

Improvement metric. The proposed OD algorithm with divided approximation is shown in

Table 3.3.

The combination of operand decomposition with the proposed approximation equations,

reduces the average error percent to less than 0.2%. In the next section we combine operand
decomposition with error correction based approaches namely the table of correction values

(TCV) and Mitchell’s error correction (MEC) term based approach.
3.4 Combining OD with Error Correction Based Approaches

The operand decomposition is combined with error correction approaches without any change. In other words, both the decomposed multiplications use similar correction tables or equations.

3.4.1 OD with Table of Correction Values (TCV)

The table of correction values is a simulation based approach. Here a constant list of values to be added to the Mitchell’s logarithmic summation is stored in a table. The fractional value of the logarithms is used to access the entries in the table. Table 3.4.1 illustrates the method to access the appropriate correction term from the table given the fraction values. In this thesis, we produced a new set of correction values for the entries, depending on the ratio of decomposed products $A \times B$ and $C \times D$. The same table of correction values is used for both the multiplications $A \times B$ and $C \times D$. The fraction values $m_a, m_b, m_c$ and $m_d$ are used to access the table, which has 64 entries. The entries in the table are developed using extensive trial and error simulations. The combination of operand decomposition with table of correction values (OD-TCV) is shown in Table 3.4.1.

The algorithm is similar to Mitchell’s calculation, expect a correction (TCV) value is added during the summation of the logarithms. Table 3.4.1 refers to the table of correction values used in this thesis. The approach produces a very good error range and decreases the average error percentage to about 0.006%. In the next section, we present the combination of operand decomposition with Mitchell’s error correction term based algorithm.

3.4.2 Combining OD with Mitchell’s Correction Term

Mitchell in this paper [17], presented a simple correction approach requiring only two extra additions. The approach adds a correction term to the original product, generated by the Mitchell’s algorithm. In this thesis we combine the Operand decomposition approach with Mitchell’s correction without any modification to the host approach. The equations
Table 3.3. OD-TCV Logarithmic Multiplication Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Decompose the numbers X and Y into A, B, C and D;</td>
</tr>
</tbody>
</table>
| 2    | Calculate $k_a \leftarrow$ leading one position of number A;  
%$k_a$ and $k_b$ are the characteristic of the logarithms of A and B; |
| 3    | Calculate $k_b \leftarrow$ leading one position of number B;  
%$m_a$ and $m_c$ refers to the mantissa part of the logarithm of A and B; |
| 4    | Calculate $m_a \leftarrow$ remaining part of operand A;  
Calculate $m_b \leftarrow$ remaining part of operand A; |
| 5    | Append as floating point numbers $f_a \leftarrow (k_a,m_a)$ and $f_b \leftarrow (k_b,m_b)$;  
% add the logarithm values and then calculate the antilogarithm; |
| 6    | $f_{ab} \leftarrow f_a + f_b + TCV(m_a,m_b)$;  
Calculate $P_{ab} \leftarrow$ antilog($f_{ab}$); |
| 7    | Perform steps 2 and 3 for decomposed multiplicands C and D and store  
the values in $k_c$ and $k_d$;  
Calculate $m_c$ and $m_d$ as in steps 4 and 5.; |
| 8    | Append $f_c \leftarrow (k_c,m_c)$ and $f_d \leftarrow (k_d,m_d)$ as a number;  
$f_{cd} \leftarrow f_c + f_d + TCV(m_c,m_d)$;  
Calculate $P_{cd} \leftarrow$ antilog($f_{cd}$); |
| 9    | % Adding the result of the decomposed multiplications ;  
Product: $X \times Y \leftarrow P_{ab} + P_{cd}$; |
| 10   | TCV($m_1,m_2$);  
% accessing correction values from TCV |
| 11   | $\text{temp}_i \leftarrow -1$;  
$\text{temp}_j \leftarrow -1$; |
| 12   | :while ($\text{temp}_i < 7$ and $m_1 > 0$)  
$m_1 \leftarrow m_1 - 0.125$;  
$\text{temp}_i \leftarrow \text{temp}_i - 0.125$; |
| 13   | :while ($\text{temp}_j < 7$ and $m_2 > 0$)  
$m_1 \leftarrow m_2 - 0.125$;  
$\text{temp}_j \leftarrow \text{temp}_j - 0.125$; |
| 14   | :if ($\text{temp}_i == -1$)  
$\text{temp}_i \leftarrow 0$;  
:if ($\text{temp}_j == -1$)  
$\text{temp}_j \leftarrow 0$; |
| 15   | :return array[\text{temp}_i][\text{temp}_j]; |

26
Table 3.4. Estimated OD-TCV Values

<table>
<thead>
<tr>
<th></th>
<th>0.125</th>
<th>0.125</th>
<th>0.25</th>
<th>0.375</th>
<th>0.5</th>
<th>0.625</th>
<th>0.75</th>
<th>0.875</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00-0.125</td>
<td>-0.0016</td>
<td>0.0076</td>
<td>0.0124</td>
<td>0.0190</td>
<td>0.0237</td>
<td>0.0278</td>
<td>0.0313</td>
<td>0.0322</td>
<td></td>
</tr>
<tr>
<td>0.125-0.25</td>
<td>0.0067</td>
<td>0.0296</td>
<td>0.0484</td>
<td>0.0641</td>
<td>0.0774</td>
<td>0.0888</td>
<td>0.0846</td>
<td>0.0274</td>
<td></td>
</tr>
<tr>
<td>0.25-0.375</td>
<td>0.0124</td>
<td>0.0484</td>
<td>0.073</td>
<td>0.1016</td>
<td>0.1223</td>
<td>0.1360</td>
<td>0.0737</td>
<td>0.0186</td>
<td></td>
</tr>
<tr>
<td>0.375-0.50</td>
<td>0.0190</td>
<td>0.0641</td>
<td>0.1016</td>
<td>0.1333</td>
<td>0.1463</td>
<td>0.1629</td>
<td>0.0337</td>
<td>0.0123</td>
<td></td>
</tr>
<tr>
<td>0.5-0.625</td>
<td>0.0237</td>
<td>0.0774</td>
<td>0.1223</td>
<td>0.1463</td>
<td>0.1122</td>
<td>0.0710</td>
<td>0.0334</td>
<td>0.0069</td>
<td></td>
</tr>
<tr>
<td>0.625-0.75</td>
<td>0.0278</td>
<td>0.0898</td>
<td>0.1260</td>
<td>0.1029</td>
<td>0.0710</td>
<td>0.0443</td>
<td>0.0218</td>
<td>0.0024</td>
<td></td>
</tr>
<tr>
<td>0.75-0.875</td>
<td>0.0313</td>
<td>0.0846</td>
<td>0.0737</td>
<td>0.0537</td>
<td>0.0364</td>
<td>0.0218</td>
<td>0.0003</td>
<td>-0.0017</td>
<td></td>
</tr>
<tr>
<td>0.875-1.00</td>
<td>0.0202</td>
<td>0.0264</td>
<td>0.0186</td>
<td>0.0123</td>
<td>0.0069</td>
<td>0.0024</td>
<td>-0.0017</td>
<td>-0.0049</td>
<td></td>
</tr>
</tbody>
</table>

for these correction terms are given in Chapter 2. This method significantly improves the error range as 99% of the logarithm multiplications now has an error value of less than 1%.

3.5 Summary

In this chapter, we presented the operand decomposition approach for improving accuracy in logarithmic multiplication. The operand decomposition was originally used by the authors in [11], to reduce the switching transitions in binary multiplication. Further, we also proposed algorithms to combine operand decomposition to previous error correction approaches to Mitchell’s logarithmic multiplication. It is seen that all the combination improves the average error percentage and the error range of the logarithmic multiplication. Experimental results confirming these are presented in the next chapter.
CHAPTER 4

EXPERIMENTAL RESULTS

This chapter discusses how the proposed algorithms are implemented and presents the compiled experimental results. The algorithms are implemented in C programming environment and compiled using the gcc compiler under Sun Solaris. The algorithms were tested with random input data generated using the rand() function in stdio.h. In the domain of logarithmic multiplication two metrics for comparison are popular, the maximum possible error and the average error percentage. Simulation results in previous correction approaches include one of these two. The proposed operand decomposition approach does not reduce the maximum possible error except for certain methods. The operand decomposition approach gives the error curve a Gaussian shape and hence, improves on the average percent error. This average error percentage is important in many digital signal processing applications including, 2D-Correlation, FIR filtering, IIR filtering and many other applications involving convolution of two matrices. The average error percent (AEP) is defined as follows:

$$Error\ Percent(EP) = \frac{TV - LV}{TV} \times 100$$  \hspace{1cm} (4.1)

where TV refers to the true value obtained using binary multiplication and LV refers to value obtained using proposed logarithmic multiplication.

$$Average\ Error\ Percent\ (AEP) = \frac{\sum_{i=1}^{N} EP_i}{N}$$  \hspace{1cm} (4.2)

where N is the number of multiplications performed. The generated results are then compared with the error percentage of the previous approaches. These methods are also implemented and applied with the same input patterns for the sake of fairness. Secondly, the
Table 4.1. Mitchell's Vs Operand Decomposed (OD)-Mitchell's: Error Percentage Level

<table>
<thead>
<tr>
<th>Error Percentage</th>
<th>Mitchell's Algorithm</th>
<th>OD-Mitchell's Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>= 0%</td>
<td>0.05</td>
<td>0.27</td>
</tr>
<tr>
<td>&lt; 1%</td>
<td>21</td>
<td>45</td>
</tr>
<tr>
<td>&lt; 2%</td>
<td>34</td>
<td>63</td>
</tr>
<tr>
<td>&lt; 3%</td>
<td>46</td>
<td>74</td>
</tr>
</tbody>
</table>

Table 4.2. Mitchell's Vs Operand Decomposed(OD)-Mitchell's: Average Error Percent (AEP)

<table>
<thead>
<tr>
<th>Input width</th>
<th>Number of Multiplications</th>
<th>Mitchell's (AEP)</th>
<th>OD-Mitchell's (AEP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1000</td>
<td>3.77</td>
<td>2.07</td>
</tr>
<tr>
<td>8</td>
<td>100000</td>
<td>3.64</td>
<td>2.12</td>
</tr>
<tr>
<td>16</td>
<td>1000</td>
<td>3.83</td>
<td>2.07</td>
</tr>
<tr>
<td>16</td>
<td>100000</td>
<td>3.85</td>
<td>2.17</td>
</tr>
<tr>
<td>32</td>
<td>1000</td>
<td>3.86</td>
<td>2.15</td>
</tr>
<tr>
<td>32</td>
<td>100000</td>
<td>3.87</td>
<td>2.17</td>
</tr>
</tbody>
</table>

thesis also gives a distribution of the number of values below a particular error value. For example in the proposed operand decomposition approach 45 percentage of its multiplications have an error of less than 1 percent, compared to only 21 percentage in Mitchell’s algorithm. Similar distribution is presented for all the operand decomposition combined error correction approaches. The proposed operand decomposition approach gives better results even with this metric in all cases. These simulation results are presented in the next section.

4.1 Operand Decomposition versus Mitchell’s Logarithm

Mitchell’s Original method due to its piecewise linear approximation in power of two intervals had an average error of 3.88%. On combination of Mitchell’s logarithm with operand decomposition the average error percentage reduces to 2.1% for 32 bit width multiplicands. The number of multiplications performed is varied from a 100 to 100000. Figures 4.1 and 4.2 shows the error spread of 500 random input multiplications for Mitchell’s and OD-Mitchell’s
Table 4.3. Divided Approximation Vs OD-DA: Error Percentage Level

<table>
<thead>
<tr>
<th>Error Percentage</th>
<th>DA</th>
<th>OD-DA</th>
</tr>
</thead>
<tbody>
<tr>
<td>= 0%</td>
<td>10</td>
<td>23</td>
</tr>
<tr>
<td>&lt; 1%</td>
<td>31</td>
<td>67</td>
</tr>
<tr>
<td>&lt; 2%</td>
<td>54</td>
<td>88</td>
</tr>
<tr>
<td>&lt; 3%</td>
<td>67</td>
<td>97</td>
</tr>
</tbody>
</table>

multiplication for 32-bit width inputs. The dots in the figures represent the error percent for individual multiplication and the curve represents the moving average over different random blocks. The same convention is followed in all the error spread graphs presented in this chapter. It can be clearly seen that for OD-Mitchell’s method more errors are near zero. Further the percentage of values below a certain error range for 100000 multiplications are shown in Table 4.1.

Secondly, the average error percentage for various input width and different N (number of multiplications is shown in Table 4.2. The table shows the average error percent for four different multiplicand widths and two different number (N) of multiplications. It can be clearly seen from these tables that the proposed work improves the error range and the average error percentage of Mitchell’s logarithmic multiplication by 45%.

4.2 Combination of OD with Other Correction Approaches

In this section, we will present the results of combining the proposed operand decomposition with previously published error correction approaches to logarithmic multiplication.

4.2.1 Divided Approximation

The operand decomposition (OD) when combined with previous error correction approaches improves the average error percentage and also the percent error range. Below, we show the error spread of 500 random input multiplications for divided approximation
Table 4.4. Divided Approximation Vs OD-DA: Average Error Percent (AEP)

<table>
<thead>
<tr>
<th>Legend : DA = Divided Approximation</th>
<th>Legend : OD = Operand Decomposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input width</td>
<td>Number of Multiplications</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>8</td>
<td>100000</td>
</tr>
<tr>
<td>16</td>
<td>1000</td>
</tr>
<tr>
<td>16</td>
<td>100000</td>
</tr>
<tr>
<td>32</td>
<td>1000</td>
</tr>
<tr>
<td>32</td>
<td>100000</td>
</tr>
</tbody>
</table>

and OD-divided approximation methods in Figures 4.3 and 4.4 respectively. In this case, the errors are more evenly distributed above and below zero, hence giving the error curve a Gaussian shape. The average percent error also achieves good improvement because of this. The coefficients for the linear equations in divided approximation were suitably selected to reduce the average percent error and the percent error range optimally. Table 4.3 show the percentage of errors below a certain value for divided approximation. The average error percentage for the divided approximation is shown in Table 4.4.

Once again, the operand decomposition method shows improvement in both the error metric calculations.

4.2.2 Correction Term Based Methods

Here we present the error percentage results of the correction term based approaches. Suitable values for the table of correction values are generated and suitably added to decrease the average error percentage. Secondly, the Mitchell’s error correction term (MEC) method is combined with operand decomposition without any modification. The results on error spread for table of correction values and OD-table of correction values are shown in Figures 4.5 and 4.6 respectively. The Mitchell’s error correction term based approach has the best error range and is mostly positive. Figures 4.7 and 4.8 show the error spread for Mitchell’s correction term (MEC) and OD-Mitchell’s correction term. Finally, the error percent range and the average error percentage for these two correction term based
approaches are shown in tables 4.5 and 4.6 respectively. In the next section, we evaluate the proposed operand decomposition approach on a image processing filter.

4.3 Gaussian Smoothing

The Gaussian smoothing operation, is a two dimensional convolution based filter, that is used to blur images. The gaussian operation is similar to mean filter, but it uses a kernel matrix that represents the shape of a Gaussian (bell-shaped hump), which detects edges more accurately.

A circularly symmetric gaussian distribution in two dimensions has the form,

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2 + y^2}{2\sigma^2}}$$  \hspace{1cm} (4.3)

where, $\sigma$ is the standard deviation of the distribution. In theory, the gaussian distribution is a continuous one and would require a large convolution kernel. However, it has been shown in practice that three standard deviations from the mean effectively approximates the gaussian distribution. An example convolution kernel matrix for gaussian distribution with a standard deviation ($\sigma$) of 1.0 is shown in Figure 4.9. Once a suitable kernel has been calculated, the gaussian smoothing operation is performed using standard convolution methods. The gaussian filter is also useful in some computational biology applications, as some cells in the visual pathways of the brain have an approximate gaussian response. Next, we present error percentage results on gaussian smoothing operation with the operand decomposed mitchell algorithm based multiplication for convolution.

4.3.1 Operand Decomposed Gaussian Smoothing

In this section, we perform the gaussian smoothing operation on a random image. Figure 4.10 shows the base image obtained from Heriot-watt university [26] for evaluating operand decomposition approach. The base image in figure 4.10 is corrupted with salt and pepper noise (drop out noise) as shown in figure 4.11, with a probability of 1% (i.e. individual
Table 4.5. Correction Term Vs OD-Correction Term: Error Percentage Level

<table>
<thead>
<tr>
<th>Error Percentage</th>
<th>TCV</th>
<th>OD-TCV</th>
<th>MEC</th>
<th>OD-MEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>= 0%</td>
<td>28</td>
<td>35</td>
<td>0.5</td>
<td>15</td>
</tr>
<tr>
<td>&lt; 1%</td>
<td>67</td>
<td>81</td>
<td>99</td>
<td>99.7</td>
</tr>
<tr>
<td>&lt; 2%</td>
<td>79</td>
<td>92</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>&lt; 3%</td>
<td>86</td>
<td>96</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 4.6. Correction Term Vs OD-Correction Term: Average Error Percent (AEP)

<table>
<thead>
<tr>
<th>Input width</th>
<th>N</th>
<th>TCV (AEP)</th>
<th>OD-TCV (AEP)</th>
<th>MEC (AEP)</th>
<th>OD-MEC (AEP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1000</td>
<td>0.200</td>
<td>-0.22</td>
<td>0.12</td>
<td>0.107</td>
</tr>
<tr>
<td>8</td>
<td>100000</td>
<td>0.280</td>
<td>-0.12</td>
<td>0.13</td>
<td>0.115</td>
</tr>
<tr>
<td>16</td>
<td>1000</td>
<td>0.391</td>
<td>0.169</td>
<td>0.14</td>
<td>0.044</td>
</tr>
<tr>
<td>16</td>
<td>100000</td>
<td>0.375</td>
<td>0.005</td>
<td>0.14</td>
<td>0.047</td>
</tr>
<tr>
<td>32</td>
<td>1000</td>
<td>0.430</td>
<td>0.039</td>
<td>0.15</td>
<td>0.042</td>
</tr>
<tr>
<td>32</td>
<td>100000</td>
<td>0.365</td>
<td>0.012</td>
<td>0.14</td>
<td>0.039</td>
</tr>
</tbody>
</table>

Table 4.7. Gaussian Smoothing: Error Comparison

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean Square Error</th>
<th>Average Error Percent (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>MA</td>
<td>21.30</td>
<td>3.20</td>
</tr>
<tr>
<td>OD-MA</td>
<td>10.62</td>
<td>2.05</td>
</tr>
<tr>
<td>DA-MA</td>
<td>13.68</td>
<td>3.39</td>
</tr>
<tr>
<td>OD-DA-MA</td>
<td>1.29</td>
<td>0.54</td>
</tr>
</tbody>
</table>
Figure 4.1. Error Spread of Mitchell’s Algorithm

Figure 4.2. Error Spread of Operand Decomposed-Mitchell

Figure 4.3. Error Spread of Divided Approximation
Figure 4.4. Error Spread of Operand Decomposed - DA

Figure 4.5. Error Spread of Table of Correction Values

Figure 4.6. Error Spread of Operand Decomposed - TCV
Figure 4.7. Error Spread of Mitchell’s Error Correction Term

Figure 4.8. Error Spread of Operand Decomposed - MEC

Figure 4.9. Discrete Approximation to Gaussian Distribution with standard deviation ($\sigma$) = 1.0
bits have been flipped with probability 1%). Salt and pepper noise is very common during transmission. The gaussian smoothing operation (blurring) is performed on this corrupted image using the kernel matrix shown in figure 4.9. The multiplications for the convolution operation during smoothing is performed using a standard array multiplier, Mitchell’s algorithm and OD-Mitchell’s algorithm. The resultant images of the smoothing operations are shown in figures 4.12 , 4.13 and 4.14 respectively. Here, we use mean square error (MSE) and average error percent (AEP) for showing the effectiveness of operand decomposition in the gaussian smoothing operations. The error comparison for the proposed approach is shown in Table 4.7. It can be inferred from table 4.7, that the proposed operand decomposition approach clearly improves mean square error and the average error percentage for the gaussian smoothing operation. The OD-Mitchell algorithm when compared to Mitchell’s Algorithm, improves the error percentage by more than 45%. Similarly the operand decomposition’s combination with divided approximation also improve the accuracy in similar levels.

![Figure 4.10. Base Image](image)

### 4.4 Summary

In this chapter, we presented the implementation methodology and the simulation results of the proposed operand decomposition approach for improving accuracy in logarithmic multiplication. The operand decomposition approach improves both the error range percent
and the average error percent in all cases. The combination of operand decomposition with table of correction value has the best average error percentage. The combination of operand decomposition with Mitchell’s error correction term (MEC) gives the best percent error range. In the next chapter, we conclude this research work.
Figure 4.13. Gaussian Smoothing with Mitchell’s Algorithm

Figure 4.14. Gaussian Smoothing with OD-Mitchell Algorithm
CHAPTER 5
CONCLUSION

Mitchell’s algorithm [17] based logarithm multiplication is desirable for digital signal processing applications, due to its low overhead property. However the piecewise straight line approximation in Mitchell’s algorithm has a high error percentage. In this thesis, we propose a novel approach to improve the accuracy in Mitchell based logarithmic multiplication using operand decomposition. A slightly modified operand decomposition was used in [11], to reduce the switching transitions in binary multiplication. The operand decomposition approach improves the average error percentage and the error range of Mitchell algorithms by around 45%.

The proposed operand decomposition approach also couples well with previously proposed error correction improvements to Mitchell’s logarithm. Efficient algorithms are developed as a part of this thesis, to optimally combine operand decomposition with divided approximation, table of correction values and Mitchell’s error correction equations method. New powers of two coefficients are developed for correction equations in divided approximation. Similarly, we also develop new set of correction values for the table of correction values approach for minimizing error range and average error percentage of the multiplication algorithm. All the combined approaches improve both the average error percent and the error range of the previously proposed error correction improvements to Mitchell’s logarithmic multiplication.
REFERENCES


41


[18] Roberto Muscedere. Difficult operations in the multidimensional logarithmic number

techniques for binary-to-multidigit multidimensional logarithmic number system
conversion using range-addressable look-up tables. pages 257–272. IEEE transactions
on computer, 2005.


[21] S. Ramaswamy and R. Siferd. "cmos vlsi implementation of a digital logarithmic mul-
Conference, 1996.


Transactions on Computers, dec, 1975.


42