LUND UNIVERSITY

DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY, FACULTY OF ENGINEERING, LTH

MASTER OF SCIENCE THESIS

Hardware Implementation of the Logarithm Function

using Improved Parabolic Synthesis

Author:

Jingou Lai

Supervisor:

Peter Nilsson
Erik Hertz
Rakesh Gangarajaiah

Lund September 2, 2013

The Department of Electrical and Information Technology
Lund University
Box 118, S-221 00 LUND
SWEDEN

This thesis is set in Computer Modern 11pt,
with the LaTeX Documentation System

Abstract

This thesis presents a design that approximates the fractional part of the based two logarithm function by using Improved Parabolic Synthesis including its CMOS VLSI implementations. Improved Parabolic Synthesis is a novel methodology in favor of implementing unary functions e.g. trigonometric, logarithm, square root etc. in hardware. It is an evolved approach from Parabolic Synthesis by combining it with Second-Degree Interpolation. In the thesis, the design explores a simple and parallel architecture for fast timing and optimizes wordlengths in computing stages for a small design. The error behavior of the design is described and characterized to meet the desired error metrics. This implementation is compared to other approaches e.g. Parabolic Synthesis and CORDIC using 65nm standard cell libraries and it is proved to have better performance in terms of smaller chip area, lower dynamic power, and shorter critical path.

Acknowledgments

This thesis would not have been possible without the support from many people.

I would like to express my deepest gratitude to the supervisors, Peter Nilsson, Erik Hertz, and Rakesh Gangarajaiah who patiently, inspiringly guided me through this thesis work.

My special thanks would go to my families and dearest friends who always have faith in me and encourage me.

Finally, my appreciation would extend to my SoC classmates for all the bright ideas and generous helps.

Jingou Lai
Lund, September 2013

Contents

List of Tables

List of Figures

List of Acronyms

ASIC	Application Specific Integrated Circuit
CORDIC	Coordinated ROtation DIgital Computer
DSP	Digital Signal Processor or Digital Signal Processing
EDA	Electronic Design Automation
FPGA	Field Programmable Gate Array
GDS	Graphic Database System
LUT	Look Up Table
MOS	Metal Oxide Semiconductor
RMS	Root Mean Square
SDF	Standard Delay Format
SPEF	Standard Parasitic Exchange format
VCD	Value Change Dump
VHDL	Very high speed integrated circuit Hardware Discription Language
VLSI	Very Large Scale Integrated circuit
VT	Threshold Voltage

Chapter 1

Introduction

Binary logarithm is widely applied in the field of graphical processing, communication systems, etc. It can substitute some high complex operation e.g. multiplication and division [1]. For high speed processing, it is not efficient to only rely on software solutions. Instead, due to the rapidly decreasing scale of Metal Oxide Semiconductor Transistor (MOS) transistors, realizing the function in Very Large Scale Integrated Circuit (VLSI) becomes applicable nowadays.

Methodologies to approach a binary logarithm in hardware, e.g. a rudimentary implementation, simply employs a direct Look-Up Table (LUT) [2] [3]. However, the large table size will be problematic both for large number of input patterns and high precision. An alternative method, polynomial approximation, can reduce design to some extent but far from enough due to its high computational strength architecture [4]. To address those problems, the Coordinate Rotation Digital Computer (CORDIC), an algorithm offers a time-multiplexed architecture, is often used [5] [6]. However it has a drawback since it has a long processing delay due to the iterative characteristic. Moreover, in consideration of error behavior, since CORDIC calculates output by configuring bit by bit, it leads errors statics to be unbalanced referring to zero [7].

An innovative methodology, parabolic synthesis, recently proposed by Erik Hertz and Peter Nilsson, suggests new hardware architectures to approximate unary functions e.g. sine, logarithm, exponential, and square root etc. [8] [9] [10]. This methodology instructs to develop and recombine several parabolic functions, which are called subfunctions, to approximate each normalized function, called original

function. The idea explores parallelism to achieve fast speed and uses simple hardware for each subfunction to reduce the overall area. Its evolved methodology, Improved Parabolic Synthesis, proposed by Erik Hertz and Peter Nilsson, combining Parabolic Synthesis and second-degree interpolation, employs only two subfunctions to approximate the original function. Accuracy is dependent on both the first subfunction and the number of intervals in the second subfunction. Thereby, the first subfunction can be chosen for less hardware complexity while the second subfunction in each subinterval is developed.

The error behavior is also an important factor to character the design. The desired error behavior is symmetric and concentrating around 0. The coefficients of the second subfunction are optimized to characterize error behavior.

A fractional part of binary logarithm function had been approximated using Parabolic Synthesis by Peyman P. The results were compared to CORDIC on a Field Programmable Gate Arrary (FPGA) and an Application Specific Integrated Circuit (ASIC). It is proved to be faster and consume less power than the CORDIC [11]. In the thesis, a binary logarithm that calculates 15 bits output precision from inputs with 14 bits mantissa range from integer 1 to 2 is implemented using the Improved Parabolic Synthesis. It results in a simpler hardware due to the advantage of the Improved Parabolic Synthesis and optimized word length in each stage. In addition, refined coefficients result in a characterized error behavior with symmetry concentrated around zero.

For implementation, it includes three hierarchical stages in the Parabolic Synthesis and the Improved Parabolic Synthesis. For logarithm implementation, in the processing stage, a squarer unit is implemented from a hardware-effective algorithm. [12]. The design is written in Very High Speed Integrated Circuit Hardware Description Language (VHDL) using the 65nm CMOS technology with low power low V_{T}, low power high V_{T} and low power standard V_{T} cell libraries with a supply voltage 1.2V library.

As result, area and timing information from synthesis and place and route (P&R) are reported. Power and energy are estimated under different operating frequencies. Those results are compared to the Parabolic Synthesis Method and the CORDIC.

1.1 Thesis Outlines

Remaining chapters are outlined below:

Chapter 2 introduces Parabolic Synthesis theory, subfunctions developing methods.

Chapter 3 explains the Improved Parabolic Synthesis theory, subfunctions developing methods.

Chapter 4 introduces the overall and different hardware architectures of Parabolic Synthesis and Improved Parabolic Synthesis.

Chapter 5 explains the error behavior analysis and its metrics with examples.

Chapter 6 describes the implementation of binary logarithm using Improved Parabolic logarithm.

Chapter 7 lists and analyzes results of implementation including area, timing, and power. It also lists the result from physical design.

Chapter 8 concludes the advantage using Improved Parabolic Synthesis, in comparison with early method, and future improvements.

Appendix A lists the area and timing results using 3 different cell libraries, which are 1.2V LPHVT, 1.2V LPLVT, and 1.2V GPSVT. The power estimation result using 1.2V LPHVT cell library is shown as well.

Chapter 2

Parabolic Synthesis

The Parabolic Synthesis methodology is based on the calculation of several second order functions, called subfunctions, expressed as $s_1(x)$, $s_2(x)$, ..., $s_n(x)$, and recombine them to approximate the original function, $f_{org}(x)$, as defined in (2.1). Notice that when the number of subfunctions reaches infinity, the product of all the subfunctions will result in the original function, $f_{org}(x)$. The original function, $f_{org}(x)$, is the target function to be approximated.

$$f_{org}(x) = s_1(x) \cdot s_2(x) \cdot s_3(x) ... \cdot s_\infty(x) \tag{2.1}$$

For the aid of the development of the subfuctions, help functions are used. As shown in (2.2), the first help function, $f_1(x)$, is defined as the division between the original function, $f_{org}(x)$, and the first subfunction, $s_1(x)$, since the first subfunction, $s_1(x)$, is designed to approximate the original function, $f_{org}(x)$.

$$f_1(x) = \frac{f_{org}(x)}{s_1(x)} = s_2(x) \cdot s_3(x) ... \cdot s_\infty(x) \tag{2.2}$$

The second subfunction, $s_2(x)$, is designed to approach the first help function, $f_1(x)$, which is to make the overall error smaller. This will also result in a second help function, defined recursively, as shown in (2.3).

$$f_2(x) = \frac{f_1(x)}{s_2(x)} = s_3(x) \cdot s_4(x) ... \cdot s_\infty(x) \tag{2.3}$$

In general, when $n \geq 2$ the n^{th} subfunction, $s_n(x)$, is designed to approach (n-1)$^{\text{th}}$ help function, $f_{n-1}(x)$, as defined in (2.4).

$$f_n(x) = \frac{f_{n-1}(x)}{s_n(x)} = s_{n+1}(x) \cdot s_{n+2}(x) ... \cdot s_\infty(x) \tag{2.4}$$

When n increases, the amplitude of the help function, $f_n(x)$, will decrease in size. This indicates that a larger number of subfunctions will result in a higher accuracy on the output.

2.1 First Subfunction

The original function, $f_{org}(x)$, must cross two points, (0,0) and (1,1), as shown in Fig. 2.1, where a convex curve, $f_{org1}(x)$, and a concave curve, $f_{org2}(x)$ are shown.

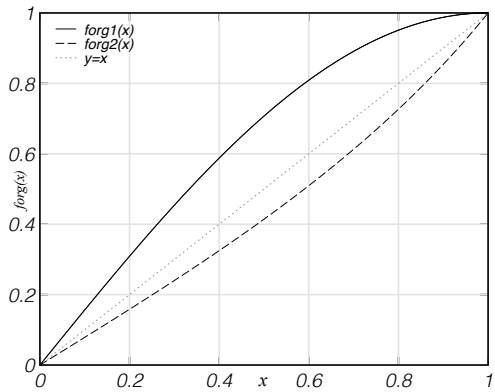

Fig. 2.1: An example of two original functions, $f_{org1}(x)$ and $f_{org2}(x)$, compared with a strait line $y = x$.

The first subfunction, $s_1(x)$, which is a second order function, is defined in (2.5).

$$s_1(x) = l_1 + k_1 x + c_1(x - x^2) \tag{2.5}$$

Since the first subfunction, $s_1(x)$, crosses (0,0), the constant part l_1 in (2.5) is calculated to be 0. The linear part k_1 in (2.5) is calculated to be 1 since the starting point is (0,0) and the end point is (1,1). The first subfunction, $s_1(x)$, is thereby reduced to (2.6).

$$s_1(x) = x + c_1(x - x^2) \tag{2.6}$$

In order to develop the first subfunction, $s_1(x)$, the original function, $f_{org}(x)$, is first divided by a strait line, $f(x) = x$. The help function, $f_{help}(x)$, is therefore defined as:

$$f_{help}(x) = \frac{f_{org}(x)}{x} \tag{2.7}$$

Applying (2.7) to the two original functions, $f_{org1}(x)$ and $f_{org2}(x)$, in Fig. 2.1, the help functions in (2.7) are calculated as the two curves in Fig. 2.2.

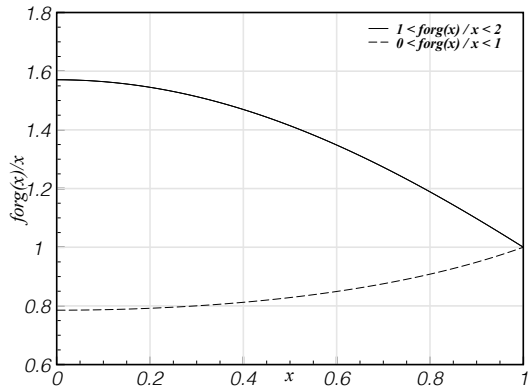

Fig. 2.2: Two help functions results from the quotient between the original functions, $f_{org1}(x)$ and $f_{org2}(x)$ in Fig. 2.1, and a strait line, $y = x$.

Additional criteria on the original function, $f_{org}(x)$, when developing c_1 are:

1. The original function, $f_{org}(x)$, must be strictly convex or concave, which means it cannot be both convex and concave.

2. The fucntion, $\frac{f_{org}(x)}{x}$, must have a limit value when x goes to 0.

3. The limited value in criteria 2 cannot be larger than 1 or smaller than -1 after subtraction by 1.

The help function, $f_{help}(x)$, in (2.7) is calculated as $1 + c_1(1 - x)$. This function cuts two points, (0,0) and (1,1). This interprets that the function starts from $1+c_1$ and ends with 1. The coefficient c_1 in the first subfunction, $s_1(x)$ is therefore

defined as in (2.8).

$$c_1 = \lim_{x \to 0} \frac{f_{org}(x)}{x} - 1 \tag{2.8}$$

2.2 Second Subfunction

The second subfunction, $s_2(x)$, is a second order function, developed to approximate the first help function, $f_1(x)$.
The first help function, $f_1(x)$, is strictly concave or convex. As the example, shown in Fig. 2.3, the first help function, $f_1(x)$, derived from the upper original function in Fig. 2.1 using (2.2).

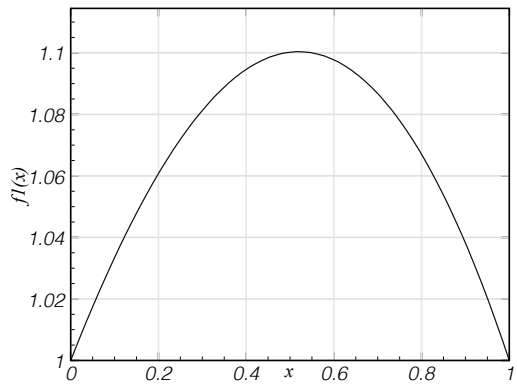

Fig. 2.3: An example of the first help function, $f_1(x)$, shown to be a strict convex curve.

The second subfunction, $s_2(x)$, is defined in (2.9).

$$s_2(x) = l_2 + k_2 x + c_2(x - x^2) \tag{2.9}$$

Since the second subfunction, $s_2(x)$, starts from the point (0,1) and ends at the point (1,1), the constant part in (2.9), l_2, is calculated as 1. The linear part in (2.9), k_2, is the gradient from (0,1) to (1,1) and therefore equals to 0.
In (2.10), the reduced second subfunction, $s_2(x)$, is shown.

$$s_2(x) = 1 + c_2(x - x^2) \tag{2.10}$$

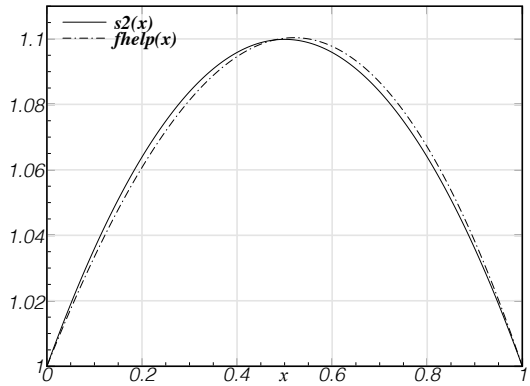

Fig. 2.4: An example of the second subfunction, $s_2(x)$, compared with first help function, $f_1(x)$.

The desired second subfunctions, $s_2(x)$, needs to cross the same points, including the start point, middle point, and end point, as the help function, shown in Fig. 2.4. To calculate c_2 in (2.9), the middle point of the first help function, $f_1(0.5)$, is used according to (2.9)

$$c_2 = \frac{f_1(0.5) - 0.5 \cdot k_2 - l_2}{0.25} = 4 \cdot (f_1(0.5) - 1) \tag{2.11}$$

2.3 Subfunction, $s_n(x)$, for $n > 2$

When further developing subfunctions, $s_n(x)$, for $n > 2$, the same methodology is applied, which is designed to approach the help function, $f_{n-1}(x)$, as stated in (2.2) and (2.4). However, all the help functions, from $f_2(x)$ to $f_n(x)$ for $n > 1$, are no more strictly convex or concave from 0 to 1 on the x axis. As an example shown in Fig. 2.5, the second help function, $f_2(x)$, results in a pair of convex and concave functions. The first function is in the interval $0 \leq x < 0.5$ and the second function is in the interval $0.5 \leq x \leq 1$.
The second help function, $f_2(x)$, is expressed in (2.12).

$$f_2(x) = \begin{cases} f_{2,0}(x) & 0 \leq x < \frac{1}{2} \\ f_{2,1}(x) & \frac{1}{2} \leq x \leq 1 \end{cases} \tag{2.12}$$

To approximate the second help function, $f_2(x)$, composed of two parabolic curves, every parabolic curve in the function is normalized into 0 to 1 in the x axis. Notice

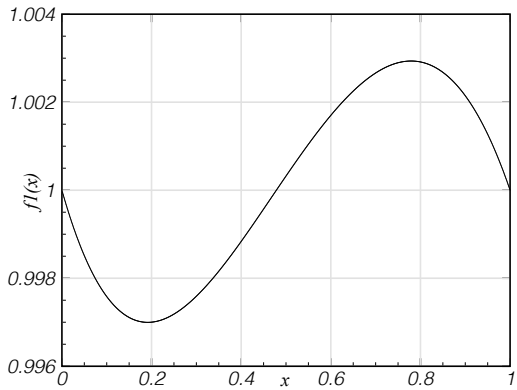

Fig. 2.5: An example of the second help function, $f_2(x)$, a pair of opposite concave and convex functions.

that x is substituted with x' in order to map the input x to the normalized parabolic curve. From x to x', the mapping is according to (2.13).

$$x' = fract(2 \cdot x) \tag{2.13}$$

Each of the parabolic curve is approximated using the method in Section 2.2 in Fig. 2.4. For the third subfunction, $s_3(x)$, when $0 \leq x < \frac{1}{2}$, $s_{3,0}(x')$ is calculated. When $\frac{1}{2} \leq x \leq 1$, $s_{3,1}(x')$ is calculated. It is expressed in (2.14).

$$s_3(x) = \begin{cases} s_{3,0}(x') & 0 \leq x < \frac{1}{2} \\ s_{3,1}(x') & \frac{1}{2} \leq x \leq 1 \end{cases} \tag{2.14}$$

In general, the n^{th} subfunction $f_n(x)$, when $n > 1$, consists of pairs of concave and convex functions, as defined in (2.15). A larger n results in a higher numbers of pairs.

$$f_n(x) = \begin{cases} f_{n,0}(x) & 0 \leq x < \frac{1}{2^{n-2}} \\ f_{n,1}(x) & \frac{1}{2^{n-2}} \leq x < \frac{2}{2^{n-2}} \\ \ldots & \\ f_{n,2^{n-1}-1}(x) & \frac{2^{n-2}-1}{2^{n-2}} \leq x < 1 \end{cases} \tag{2.15}$$

As a consequence, in general, when developing the n^{th} subfunction, $s_n(x)$, when $n > 2$, the subfunction is defined, as shown in (2.16).

$$s_n(x) = \begin{cases} s_{n,0}(x_n) & 0 \leq x < \frac{1}{2^{n-2}} \\ s_{n,1}(x_n) & \frac{1}{2^{n-2}} \leq x < \frac{2}{2^{n-2}} \\ ... & \\ s_{n,2^{n-2}-1}(x_n) & \frac{2^{n-2}-1}{2^{n-2}} \leq x < 1 \end{cases} \tag{2.16}$$

Similarly, in order to map the input to the normalized parabolic curve, the input x is substituted by x_n. In (2.17), x_n is the fractional part of the product of x and 2^{n-2}.

$$x_n = fract(2^{n-2}x) \tag{2.17}$$

Each parabolic curve in the n^{th} subfunction, $s_n(x)$, is $s_{n,m}(x_n)$. $s_{n,m}(x_n)$ is defined in (2.18),

$$s_{n,m}(x_n) = 1 + (c_{n,m} \cdot (x_n - x_n^2)) \tag{2.18}$$

where the $c_{n,m}$ for each $s_{n,m}$, is calculated similar as in Section 2.2, as defined in (2.19):

$$c_{n,m} = 4 \cdot (f_{n-1,m}(\frac{2 \cdot (m+1) - 1}{2^{n-1}}) - 1) \tag{2.19}$$

Chapter 3

Improved Parabolic Synthesis

The extended method, called Improved Parabolic Synthesis, uses only two subfunctions to approximate the original function, $f_{org}(x)$, as shown in (3.1).

$$f_{org}(x) = s_1(x) \cdot s_2(x) \tag{3.1}$$

The first subfunction, $s_1(x)$, is developed to be a parabolic function. The second subfunction, $s_2(x)$, combines parabolic functions with second-degree interpolation. In this methodology, the first subfunction, $s_1(x)$, and the second subfunction, $s_2(x)$, are developed with conformity.
In order to apply the methodology, as shown in Fig. 2.1, the original function, $f_{org}(x)$, needs to cut (0,0) and (1,1). Additional constraints applied to the original function, $f_{org}(x)$, when using the methodology are:

1. The original function, $f_{org}(x)$, must be strictly convex or concave. It cannot be both convex and concave.
2. The function, $\frac{f_{org}(x)}{x}$, must have a limit value when x goes to 0.

3.1 First Subfunction

The first subfunction, $s_1(x)$, is developed to approximate the original function, $f_{org}(x)$, similar to section 2.1, as defined in (2.6). Depending on the convexity or concavity of the original function, $f_{org}(x)$, a proper c_1 in (2.6) is chosen by

sweeping it from range 0 to 1.2 or -1.2 to 0 respectively. With every value of c_1, the first subfunction, $s_1(x)$, is developed.

3.2 Second Subfunction

The second subfunction, $s_2(x)$, is developed to approximate the first help function, $f_1(x)$, as shown in Section 2.2.

3.2.1 Second Degree Interpolation

In the Improved Parabolic Synthesis, as shown in Fig. 3.1, the first help function, $f_1(x)$, is divided into intervals of power of 2 numbers. To develop the second subfunction, $s_2(x)$, in each subinterval, the curve is normalized.

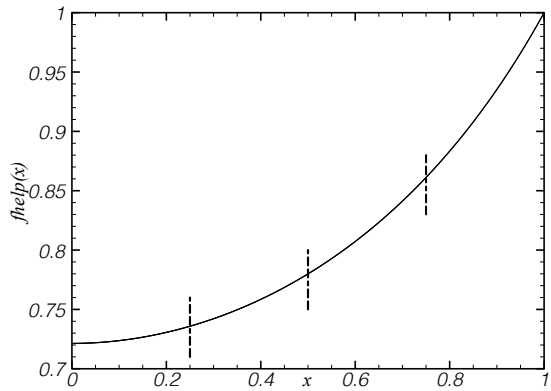

Fig. 3.1: The first help function, $f_1(x)$, divided into 4 intervals.

The general equation of the second subfunction that denotes the i^{th} interval is shown in (3.2):

$$s_{2,i}(x) = l_{2,i} + k_{2,i} \cdot x_w + c_{2,i} \cdot (x_w - x_w^2) \tag{3.2}$$

where i is the integer from 0 to number of intervals-1.

The second subfunction in each interval, $s_{2,i}(x)$, is developed to approximate the help function in that interval, $f_{1,i}(x)$, by cutting 3 points, which are the start point, middle point, and end point of the first help function in that interval, $f_{1,i}(x)$. In

general, the second subfunction, $s_{2,i}(x)$, can be expanded with all the intervals as:

$$s_2(x) = \begin{cases} s_{2,0}(x_\omega) & 0 \leq x < \frac{1}{2^\omega} \\ s_{2,1}(x_\omega) & \frac{1}{2^\omega} \leq x < \frac{2}{2^\omega} \\ ... & \\ s_{2,I-1}(x_\omega) & \frac{I-1}{2^\omega} \leq x < 1 \end{cases} \tag{3.3}$$

Note that x is substituted by x_w. The input x is multiplied by 2^w to transfer the inputs in each interval to the normalized domain, in which way the second subfunction, $s_2(x)$, approximates the first subfunction, $f_1(x)$:

$$x_\omega = fract(2^\omega x) \tag{3.4}$$

As shown in (3.5), I is the number of intervals, which is expressed in a power of 2 number. The number of intervals can be chosen as radix-2 numbers. It will further benefit when developing the hardware architecture.

$$I = 2^\omega \tag{3.5}$$

Similar to the Parabolic Synthesis, the offset of the second subfunction in the i^{th} interval, $l_{2,i}$, is simply the starting point of that interval:

$$l_2, i = f_1(start, i) \tag{3.6}$$

The ingredient of the second subfunction in the i^{th} interval, $k_{2,i}$, is the difference between the start point and the end point of that interval:

$$k_2, i = f_1(start, i) - f_1(end, i) \tag{3.7}$$

The second degree component, $c_{2,i}$, is developed by the help of middle point of each interval, shown in (3.8).

$$c_2, i = 4 \cdot (f_{1,i}(0.5) - l_{2,i} - k_{2,i} \cdot 0.5) \tag{3.8}$$

For the purpose of saving an adder in hardware, j_i is preset as in (3.9).

$$j_{2,i} = k_{2,i} + c_{2,i} \tag{3.9}$$

3.3 Develop two subfunctions concurrently

In order to find the coefficient c_1 of the first subfunction, $s_1(x)$, the following steps are performed:

1. Choose a value of the coefficient c_1. With every value of the coefficient c_1, the first subfunction, $s_1(x)$, is developed according to (2.6).
2. Since the first subfunction, $s_1(x)$, is set, the second subfunction, $s_2(x)$, is developed according to (3.1).
3. Since the first subfunction, $s_1(x)$, and the second subfunction, $s_2(x)$, have been developed, the output precision is calculated.

By Changing the value of the coefficient c_1 and repeat Step 1 to 3, the output precision is therefore a function of the coefficient c_1. For the explanation of the Step 3, the output precision is calculated by the error function. The error function, $e(x)$, is define in (3.10):

$$e(x) = |s_1(x) \cdot s_2(x) - f_{org}(x)| \tag{3.10}$$

It results in the differences between recombined result, $s_1(x) \cdot s_2(x)$, and original function, $f_{org}(x)$, expressed in decibel(dB) unit. Since the 1 bit precision in the floating point, namely 0.5, is approximately -6 dB, as shown in (3.11),

$$20 \cdot log_{10}(0.5) \approx 20(-0.301) \approx -6dB \tag{3.11}$$

the bit precision is the quotient between the maximum of the error function, $max\{e(x)\}$, in dB and -6, as shown in (3.12).

$$bit\ precision = \frac{20 \cdot log_{10}(max\{e(x)\})}{-6} \tag{3.12}$$

In contrast to the Parabolic Synthesis method, the coefficient c_1 is developed by considering overall precision on the output. The preferable c_1 is the value that fulfills the precision requirement and results in a simple architecture.

The bit precision is therefore a function of c_1, as an example of sine approximation using this approach shown in Fig. 3.2. Two peak values are detected, one is around 0.3 and the other is around 1.1.

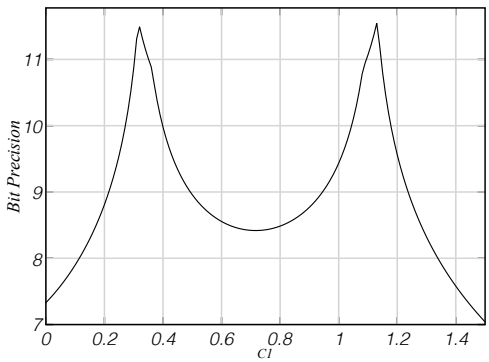

Fig. 3.2: The bit precsion as a function of c_1 with 1 interval in the second subfunction, $s_2(x)$.

After combining with the Second-Degree Interpolation, the bit precision depends not only on the coefficient c_1 but also the number of intervals. As shown in Fig. 3.3, for the higher number of intervals, the output precision curve result in higher precision.

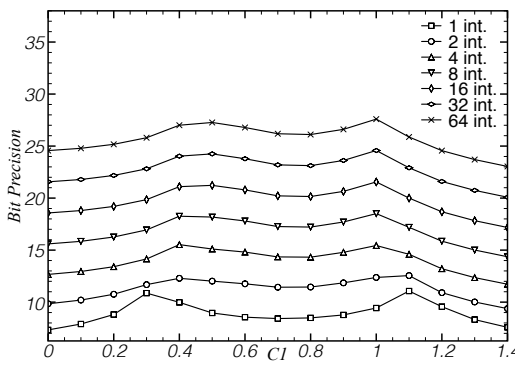

Fig. 3.3: The bit precision depending on the value of c_1 with 1, 2, 4, 8, 16, 32, 64 interval in the second subfunction, $s_2(x)$.

Typical values for c_1 are 1, 0.5, and 0, with which numbers the hardware will be

simplified.
With more intervals, freedom of choosing c_1 is increased. By choosing the typical values above, the output precision can be increased.

Chapter 4

Hardware Architecture

The hardware architecture is shown in Fig. 4.1. The architecture consists of three stages: preprocessing, processing, and postprocessing. The Preprocessing and Postprocessing stages are transformation stages, while the Processing stage approximates the original function, $f_{org}(x)$.

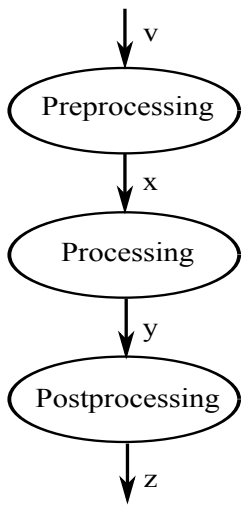

Fig. 4.1: Three stages hardware architecture for both Parabolic Synthesis and Improved Parabolic Synthesis, shown in hierarchy view.

This approach is applicable for both Parabolic Synthesis and Improved Parabolic Synthesis.

4.1 Preprocessing

In the Preprocessing stage, the input v maps the input domain to the output x into the interval from 0 to 1.
As an example, the preprocessing stage of $\sin(x)$ implementation process the input v from the input domain, which is from 0 to $\frac{\pi}{2}$, to the output x into the interval from 0 to 1 by multiplying the $\frac{2}{\pi}$.

4.2 Processing

The processing stage processes the input x that results in the output y, which is the approximated quantity from the processing stage function or the original function, $f_{org}(x)$. The parabolic Synthesis or Improved Parabolic Synthesis is applied in this stage to approximate the processing stage function, $f_{org}(x)$. As shown in Fig. 4.2, the architecture can fulfill the calculation of (2.1) and (3.1).

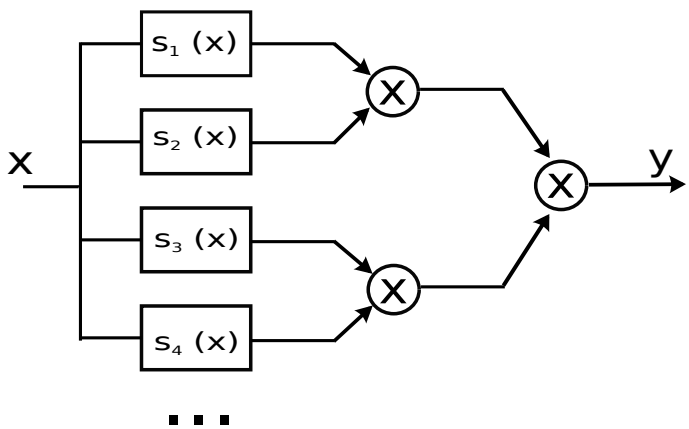

Fig. 4.2: The parallel hardware architecture for Parabolic Synthesis.

4.2.1 Architecture of Parabolic Synthesis

The unrolled architecture of (2.1) is shown in Fig. 4.3 with the subfunctions when $n = 4$.

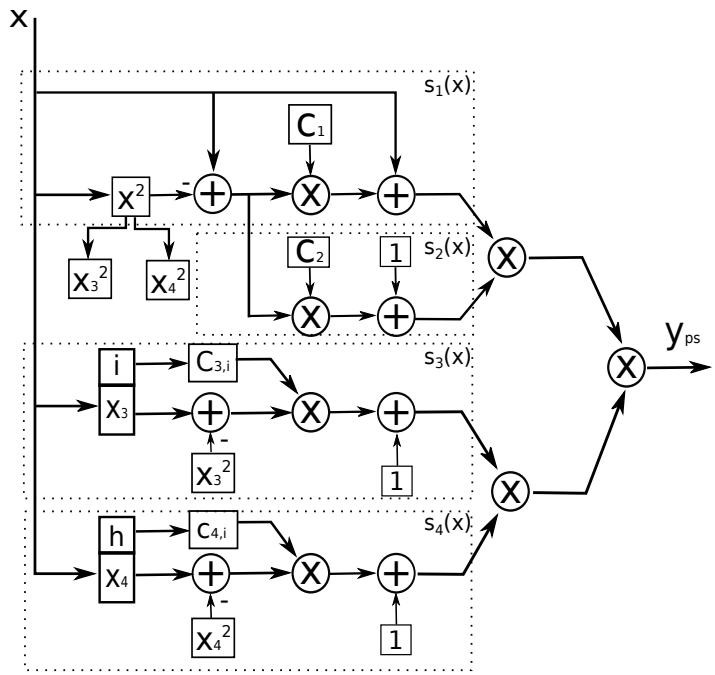

Fig. 4.3: Processing stage architecture for the Parabolic Synthesis with 4 subfunctions.

To process the first subfunction, $s_1(x)$, and the second subfunction, $s_2(x)$, x is the direct input. For the third subfunction, $s_3(x)$, x is divided into 2 parts, the interval and the input. The interval part are the most significant bits that select the step function, $c_{3,i}(x)$, and the rest of the bits is the input x_3. It is similar when computing the fourth subfunction, $s_4(x)$, except the use of 1 more significant bit as the interval part and 1 less bit for the input. The squarer unit is used to produce the the square module x^2, the partial products x_3^2 and x_4^2 for the subfunctions. Note that the architecture shows that in the first subfunction, $s_1(x)$, k_1 is equal to 1 and l_1 is equal to 0. For the n^{th} subfunctions, $s_n(x)$, that is when $n > 1$, the k_n equals to 0 and l_n equals to 1.

4.2.2 Architecture of Improved Parabolic Synthesis

The architecture of (3.1), as shown in Fig. 4.4, only computes and combines two subfunctions, $s_1(x)$ and $s_2(x)$.

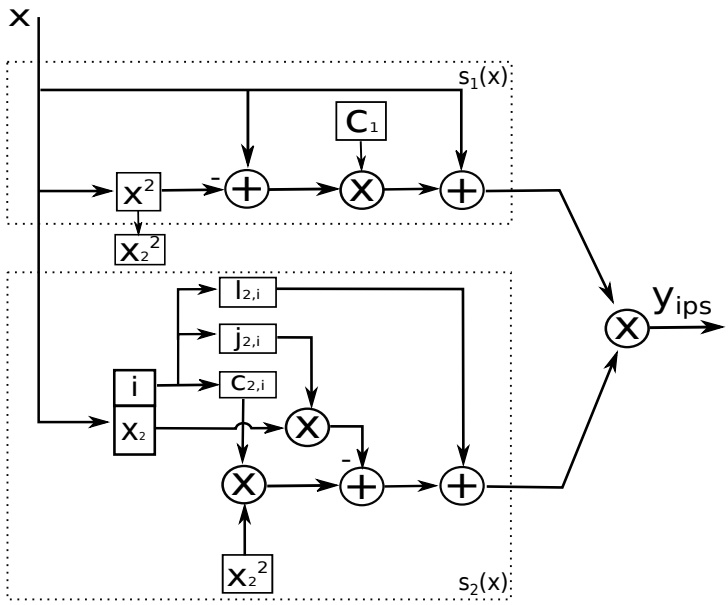

Fig. 4.4: The processing stage architecture for Improved Parabolic Synthesis.

In the second subfunction, $s_2(x)$, the most significant bits of the input x are used to select the set of coefficients, $c_{2,i}$, $k_{2,i}$, $l_{2,i}$. The number of significant bits, which is the ω in (3.5), are determined by the number of intervals I. The squarer unit produce the products of x^2 and partial products x_ω^2, which are used in the first subfunction, $s_1(x)$, and the second subfunction, $s_2(x)$, respectively.

4.2.3 Floating-Point Operations

In hardware, a floating point number is represented by a fixed point number and noted by a fractional length. For the addition(or subtraction), the operation should be performed with the alignment of the fractional length. For the multiplication, the numbers are simply multiplied and the fractional length is accumulated as the sum of fractional parts of the multiplier and the multiplicand.

With this representation system, the wordlengths of the coefficients, $c_{2,i}$ and $k_{2,i}$ can be reduced.

4.2.4 Algorithm for squarer

The algorithm of the square unit, x^2 and x_ω^2, in Fig. 4.4, can be implemented from an algorithm shown in Fig. 4.5.

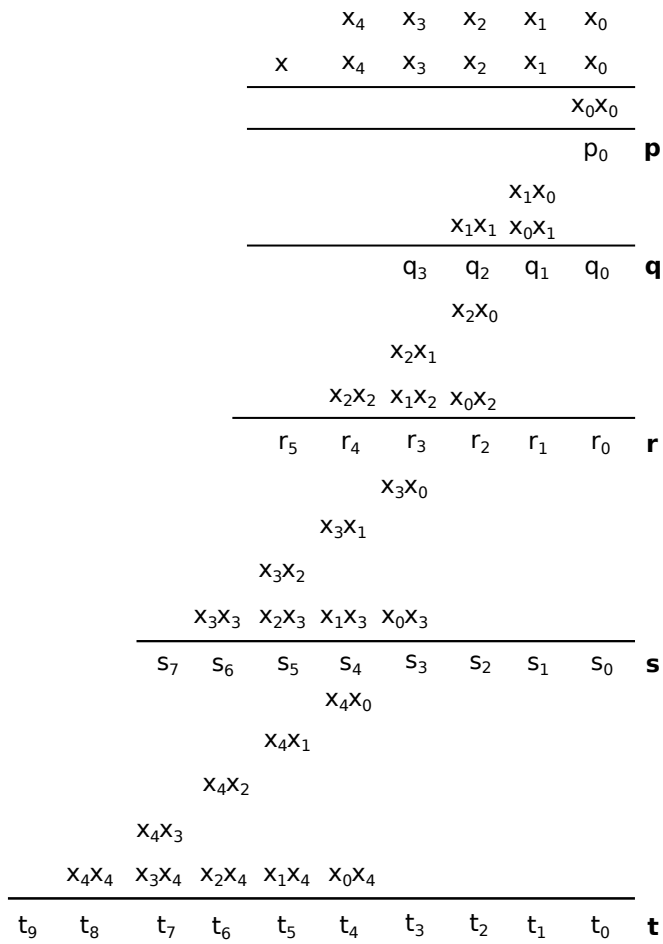

Fig. 4.5: The algorithm of the optimized architecture of the squarer unit, in which the reduced logical operations are used to calculate the partial products.

It calculates and adds the partial products p, q, r, s, and t etc. to produce the final

result. Following this algorithm, the number of partial products is implemented and controlled by the number of bits from the parameterized input x.
The advantage of implementing the squarer using this algorithm is that both the chip area and the latency of the squarer are about half compared to the corresponding multiplier.

4.2.5 Truncation and Optimization

To represent the coefficients, $c_{2,i}$, $j_{2,i}$, and $l_{2,i}$ in hardware, they are truncated to feasible binary numbers and fractional lengths as described in Section.4.2.3.
The optimization to the coefficients $c_{2,i}$, $j_{2,i}$, and $l_{2,i}$ can characterize error behavior, which will be described in Chapter 5.
The wordlengths can be reduced to some extent while the system still maintain the required output precision.
Under the condition to meet the output precision, different wordlengths between the operations have been simulated and one combination that results in a minimized architecture and a best error behavior is chosen.

4.3 Postprocessing

The Postprocessing stage processes the output y, which is the approximated result, in the range from 0 to 1, to the z with the range of the target function, to fulfill the approximation.
As the example of $\sin(x)$ approximation, the range of y is from 0 to 1, which is the same range to the target function, $\sin(x)$ with x from 0 to $\frac{\pi}{2}$. Therefore, the postprocessing stage function is $z = y$.

Chapter 5

Error Analysis

After the development of the first subfunction, $s_1(x)$, the second subfunction, $s_2(x)$, in Chapter 3, and the architecture in Chapter 4, the following step is to determine the wordlengths to be used in the architecture. This will effect the error behavior, As shown in Fig. 5.1. The black curve is the error function before truncation, and the grey is the error function after the truncation.

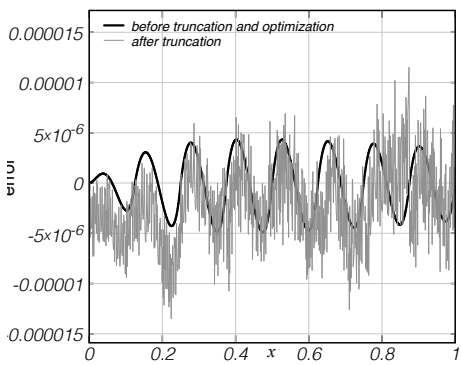

Fig. 5.1: The error functions before and after truncation.

As shown in the Fig. 5.1, after truncation, the error function has a negative offset compared to the error function before the truncation.
To neutralize this effect, the coefficients in the second subfunction, $s_2(x)$, are adjusted. It is fulfilled by manipulating those coefficients to result in a normal distributed error. This will help to reduce the wordlengths of the architecture in Fig.

4.4 since the margin between the maximum error and the required precision becomes larger. The error function after truncation is shown in Fig. 5.2, where the black curve is the error function before the truncation and optimization and the grey is the error function after the truncation and optimization.

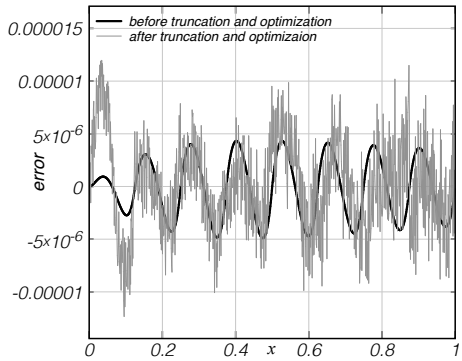

Fig. 5.2: The error functions before and after truncation and optimization.

As shown in Fig. 5.2 and 5.1, the error function after the truncation and normalization is more evenly distributed around 0 than the the error function only after truncation.

5.1 Error Behavior Metrics

When developing the architecture to result in a normal distributed error behavior, for the supplements of the error function, some metrics are important, namely the max error, the min error, the median error, the mean error, the standard deviation, and the Root Mean Square (RMS) error.

5.1.1 Maximum/Minimum Error

The maximum or minimum error is the maximum or minimum value of the error function respectively, as shown in (5.1) and (5.2).

$$e_{max} = max\{e(x)\} \tag{5.1}$$

$$e_{min} = min\{e(x)\} \tag{5.2}$$

The max or min error gives the precision bottleneck of the design.

5.1.2 Median Error

The median error gives an error value that is in the middle value of all the error samples. For an odd number of samples, it is the value makes an equal number of samples that are larger or smaller than that value. For an even number of samples, it is the mean value of central values. The median error shows the skewness of the error distribution.

5.1.3 Mean Error

The mean error is the average value of error error function, as shown in (5.3)

$$e_{mean} = \frac{1}{n}\sum_{x=0}^{n} e(x) \tag{5.3}$$

Where n is the number of samples.

5.1.4 Standard Deviation

The standard deviation is the square root of the average from the sum of the square of difference between the error and the mean error, as shown in (5.4).

$$\sigma = \sqrt{\frac{1}{n}\sum_{x=0}^{n}[e(x)-e_{mean}]^2} \tag{5.4}$$

The standard deviation indicates the dispersion around mean.

5.1.5 RMS(Root Mean Square)

The root mean square is the square root of the average from the sum of the square of errors, as shown in (5.5).

$$\sigma = \sqrt{\frac{1}{n}\sum_{x=0}^{n} e(x)^2} \tag{5.5}$$

The RMS error gives the equivalent quantity of a varying value.
In order to result in a normal distributed error, the optimization is expected to make the Standard deviation and RMS equal.

5.2 Error Distribution

Another tool to analyze the error is to use the histogram of the error function. An histogram can indicate the distribution of the error, it gives the number of the errors for all the specific values, as shown in Fig. 5.3.

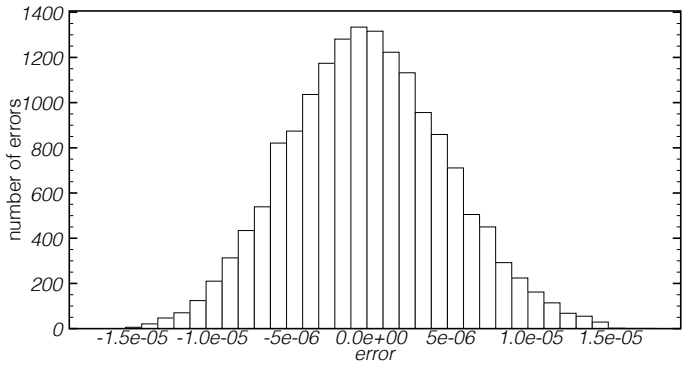

Fig. 5.3: The error histogram of the error function in Fig. 5.2.

Fig. 5.3 shows that the error histogram of Fig. 5.2 is evenly distributed around 0.

Chapter 6

Implementation of Logarithm

The based-2 logarithm function that calculates the number from 1 to 2 with 14 bits mantissa and produces the output with 15 bits precision is implemented in hardware using the improved Parabolic Synthesis methodology described in Chapter 3. Subfunctions are developed adopting the approach from Section 3.3, within which the coefficient c_1 is chosen for the simplest hardware. For the hardware implementation, the architecture and optimization in Chapter 4 is used. The error metrics in Chapter 5 is listed.

6.1 Development of Subfunctions

To derive the original function, $f_{org}(x)$, as shown in Fig. 6.1, the binary logarithm function with the input x ranging from 1 to 2 is simply shifted 1 to the left.

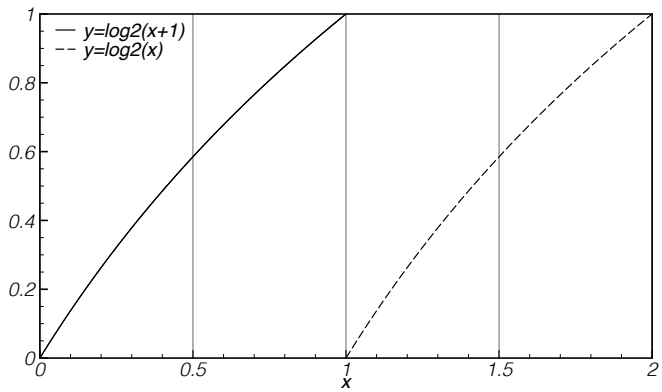

Fig. 6.1: Normalization of binary logarithm x range from 1 to 2, in which the dashed line is the function before normalization and the solid line is the original function, $f_{org}(x)$.

Therefore, the original function, $f_{org}(x)$, is:

$$f_{org}(x) = log_2(x+1) \tag{6.1}$$

6.1.1 Development of c_1

When different coefficient c_1 in the first subfunction, $s_1(x)$, combining second-degree interpolation with different number of interval in the second subfunction, $s_2(x)$, as the method described in section 3.3.2, the result is shown in Fig. 6.2. It plots the precision as a function of c_1 from 0 to 1.4 under the number of intervals from 1 to 64 for the second subfunction.

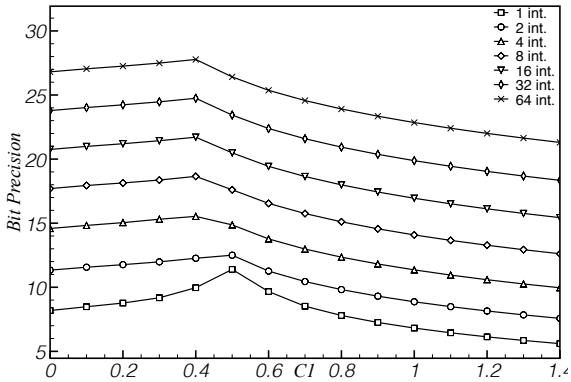

Fig. 6.2: With the interval of 1, 2, 4, 8, 16, 32, and 64, in the second subfunction, $s_2(x)$, the output precision as a function of the coefficient c_1 in the first subfunction, $s_1(x)$.

To achieve a simple hardware, the coefficient c_1 is chosen to be 0. To compensate the output precision, the number of intervals is chosen to be 8.

6.1.2 First Subfunction

Since the coefficient c_1 is set to 0, the first subfunction, $s_1(x)$ is defined in (6.2):

$$s_1(x) = x \tag{6.2}$$

As the first subfunction, $s_1(x)$, is developed, the first help function, $f_1(x)$, is defined in (6.3):

$$f_1(x) = \frac{f_{org}(x)}{s_1(x)} = \frac{log_2(x+1)}{x} \tag{6.3}$$

The first help function, $f_1(x)$, helps to develop the second subfunction, $s_2(x)$, as shown in Section 3.2.

6.1.3 Second Subfunction

To develop the second subfunction, $s_2(x)$, the methodology from section 3.2 is used. In the outcome from (3.3) to (3.9), the 8 sets of coefficients, $l_{2,i}$, $j_{2,i}$, and $c_{2,i}$ of the (3.2), after truncation and optimization, are listed in Tab. 6.1 to 6.3, respectively.

Table 6.1: the optimized 8 coefficients $l_{2,i}$ in the second subfunction, $s_2(x)$.

coefficient	Value
$l_{2,0}$	1.44268798828125000
$l_{2,1}$	1.35939788818359375
$l_{2,2}$	1.28771209716796875
$l_{2,3}$	1.22514343261718750
$l_{2,4}$	1.16991424560546875
$l_{2,5}$	1.12069702148437500
$l_{2,6}$	1.07646942138671875
$l_{2,7}$	1.03644561767578125

Table 6.2: the optimized 8 coefficients $j_{2,i}$ in the second subfunction, $s_2(x)$.

coefficient	Value
$j_{2,0}$	-0.089294433593750
$j_{2,1}$	-0.076629638671875
$j_{2,2}$	-0.066589355468750
$j_{2,3}$	-0.058471679687500
$j_{2,4}$	-0.051849365234375
$j_{2,5}$	-0.046447753906250
$j_{2,6}$	-0.041900634765625
$j_{2,7}$	-0.038085937500000

Table 6.3: the optimized 8 coefficients $c_{2,i}$ in the second subfunction, $s_2(x)$.

coefficient	Value
$c_{2,0}$	-0.0060424804687500
$c_{2,1}$	-0.0049438476562500
$c_{2,2}$	-0.0040435791015625
$c_{2,3}$	-0.0032501220703125
$c_{2,4}$	-0.0026397705078125
$c_{2,5}$	-0.0022277832031250
$c_{2,6}$	-0.0018920898437500
$c_{2,7}$	-0.0016479492187500

The effective wordlengths representing the coefficients $l_{2,i}$, $j_{2,i}$, and $c_{2,i}$ in the second subfunction, $s_2(x)$, are selected as 18, 12, and 8 bits respectively.
The optimized wordlengths between the operations will be described in Section.6.2.2.

6.2 Hardware Architecture

The architecture for the implementation of binary logarithm is shown in Fig. 6.3.

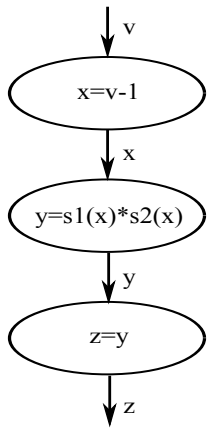

Fig. 6.3: Hardware architecture of logarithm in hierarchy

It is divided into 3 stages: preprocessing, processing, and postprocessing, as shown in Fig. 4.1. In the Preprocessing stage, it is simply subtracted by 1 from its operand. In the Processing stage, it uses the Improved Parabolic Synthesis method to approximate the original function, $log_2(x+1)$. In the Postprocessing stage, the output is directly equal to the input.

6.2.1 Preprocessing

Since the Improved Parabolic Synthesis approximates the original function, $f_{org}(x)$, in (6.1), for the binary logarithm, $log_2(v)$, with interval from 1 to 2, The input v is therefore subtracted by 1 to get x normalized from 0 to 1. The Preprocessing stage function is shown as:

$$x = v - 1 \tag{6.4}$$

In hardware, therefore the input in the next stage represents only the mantissa part of the number between 1 and 2.

6.2.2 Processing

In the Processing stage, the fractional part of logarithm is approximated using improved Parabolic Synthesis, which contains two subfunctions, where the first sub-

function is a parabolic function and the second function is a second-degree interpolation. The architecture with optimized wordlengths is shown in Fig. 6.4.

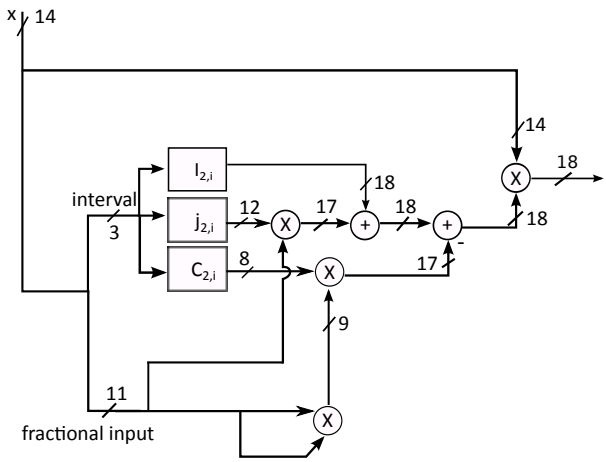

Fig. 6.4: Hardware architecture of logarithm in the processing stage

The optimization of the architecture has an impact on the error behavior, which is to be described in Section 6.3.

6.2.3 Postprocessing

Since the normalization is only the left shift on the coordinate, therefore, in the Postprocessing stage, the output is simply equal to the input, as shown in (6.5).

$$z = y \tag{6.5}$$

6.3 Error Behavior

The error behavior of the implementation is shown in Fig. 6.5, where the black curve is the error function before truncation and optimization and the grey is the error function after the truncation and optimization.

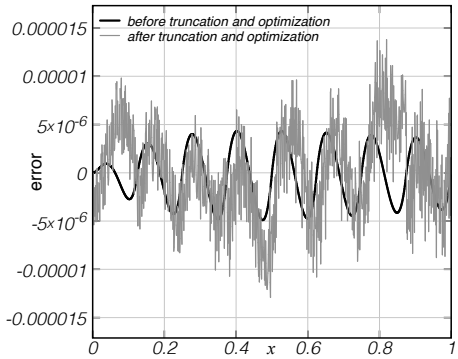

Fig. 6.5: The error function before and after the truncation and optimization.

In Fig. 6.5, it indicates that after the error function after the truncation and optimization is evenly distributed around 0.
The error function is expressed in dB unit and shown in Fig. 6.6.

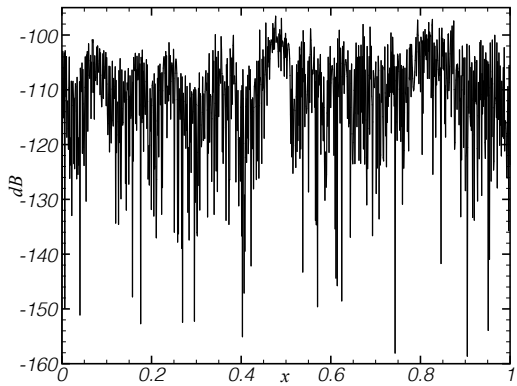

Fig. 6.6: Absolute error function expressed in dB unit of the Fig. 6.5.

As described in (3.11), since all the errors are below -90dB, the precision requirement of 15 bits is satisfied.
For the histogram of the error in Fig. 6.5, which is two-sided, symmetric and correlated to the normal distribution, shown in Fig. 6.7.
As shown in Fig. 6.7, the errors of the optimized design distributed around $-1.5 \cdot 10^{-5}$ to $1.5 \cdot 10^{-5}$, which has most of the errors around 0, and the number is grad-

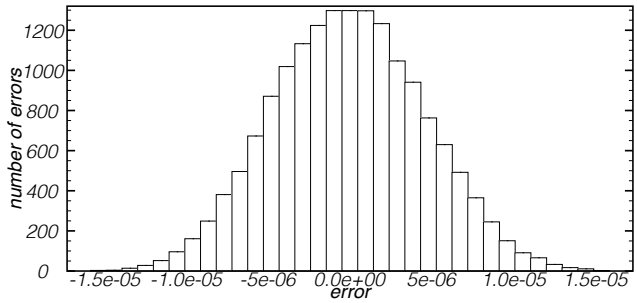

Fig. 6.7: The error histogram before the optimization on the coefficients of the second subfunction, $s_2(x)$, and the wordlength between operations.

ually decreasing to the sides.
The error metrics in Section 5.1 for the implementation are listed in Tab. 6.4.

Table 6.4: The error metrics of the truncated and optimized implementation for logarithm.

	Value	Bits
Min error	-0.000015692179454	15.96 bits
Max error	0.000015897615491	15.94 bits
Mean	0.000000019483566	25.61 bits
Median	-0.000000025005258	
Standard Deviation	0.000004737796437	
RMS	0.000004737691911	

The min and max error show a good symmetry of the error behavior. The similarity between standard deviation and RMS indicate that the error histogram is highly correlated to the normal distribution.

Chapter 7

Implementation Results

In this Chapter, the implementation results including the area, timing, and power estimation using 65nm Low Power High V_{T} (LPHVT) CMOS technology are listed and compared. A full list of results using Low Power Low V_{T} (LPLVT) and General Purpose Standard V_{T} (GPSVT) cell libraries is shown in Appendix A. The binary logarithm is also implemented using Parabolic Synthesis and CORDIC. For Parabolic Synthesis implementation, 4 subfunctions are used and no optimization to the wordlengths of design has been performed. For CORDIC implementation, 20 iterations are used, where 15+1 of 20 is used for accuracy and iteration 4, 7, 10, and 13 are used to ensure convergence [13]. Notice that no pipeline are used in any of those implementations.
The results are compared to the Improved Parabolic Synthesis approach and will be described in Chapter 8.

7.1 Area Information

The synthesis tool estimates the ASIC area for logic gates. Least area can be extracted by applying least-area constraint. As shown in Tab. 7.1, the binary logarithm possesses less than $4800\mu m^2$.

Table 7.1: ASIC synthesis result for the Improved Parabolic Synthesis when least area constraint is applied.

	Area(μm^2)
least area	4785

Under the normal conditions, where no constraints are applied to the synthesis, the results for the 3 approaches are listed in Tab. 7.2.

Table 7.2: ASIC synthesis area for the 3 methods

Desgin	Area(μm^2)
CORDIC	12893
Parabolic Synthesis	16258
Improved Parabolic Synthesis	4865

The Improved Parabolic approach possesses much less area compared to the other two methods.

7.2 Timing Information

The timing information in a design gives the bottleneck for the highest clock frequency. As shown in the Tab. 7.3, it lists the timing information when the fastest design is required.

Table 7.3: ASIC synthesis timing result for fastest constraint

constraint	timing path(ns)	frequency(Hz)
fastest design	1.71	584MHz

Under the normal constraint, Tab. 7.4 compares the timing results of the 3 implementations.

Table 7.4: ASIC synthesis timing results for two methods

Desgin	timing path(ns)	frequency(Hz)
CORDIC	86.96	11.5MHz
Parabolic Synthesis	21.03	47.5MHz
Improved Parabolic Synthesis	6.96	140MHz

As the results derive, the binary logarithm function designed in Improved Parabolic Synthesis can be implemented in a system that has a local clock frequency of 1/6.96ns = 140MHz. When using Parabolic Synthesis, it can be implemented with a clock frequency of 1/21.03ns = 47.5MHz. For the CORDIC implementation, however, due to the iterative character, the equivalent frequency is 1/86.96ns = 11.5MHz [11].

7.3 Power and Energy Estimation

7.3.1 Power analysis

The CMOS transistors consist of two power sources: dynamic power, and static power, where

$$P_{total} = P_{dynamic} + P_{static} \tag{7.1}$$

The static power source is the leakage current when the power is on. The dynamic power consists of switching power and internal power:

$$P_{dynamic} = P_{switching} + P_{internal} \tag{7.2}$$

where the switching power is the charge and discharge from transistor capacitors and the internal power source is the transition spike current when transistors are short-circuited.
The dynamic power is positive proportional to the switching activity, α, clock frequency, f, the equivalent capacitors of the transistor, C, and the supply voltage, V_{DD}.

$$P_{dynamic} = \alpha f C V_{DD}^2 \tag{7.3}$$

The switching activity, α, is estimated from Value Change Dump (VCD) file generated from simulation tools.
The power estimation of the binary logarithm using the Improved Parabolic Synthesis, Parabolic Synthesis, and CORDIC are plotted in Fig. 7.1 under different frequencies.

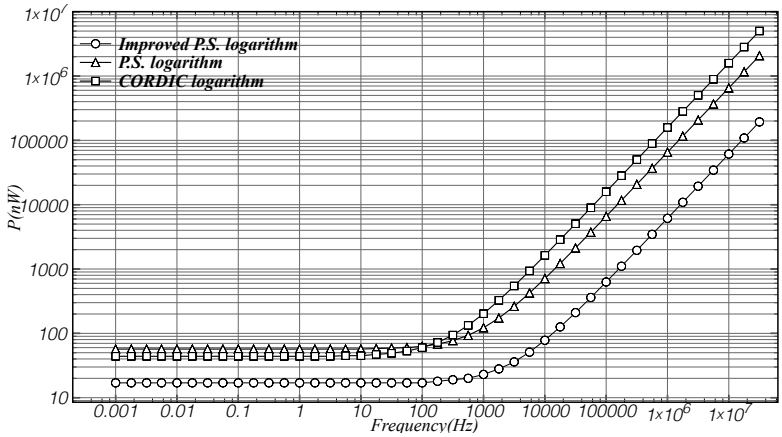

Fig. 7.1: Power estimation for 3 designs at different frequencies

For the same cell library, the static power depends on the number of transistors, which will be the dominating dissipation source during low frequencies. When frequency increases, dynamic power start to possess more and more power, which thereby become the main source of dissipation.
The binary logarithm function implemented with Improved Parabolic Synthesis, due to less area and lower switching activity, consumes much less static power and dynamic power compared to the other 2 approaches.

7.4 Physical Design

In The physical design, The Electronic Design Automation (EDA) tools combine the netlist and library files, which results in the Graphic Database System (GDSII) for fabrication. The layout of the binary logarithm is shown in Fig. 7.2.

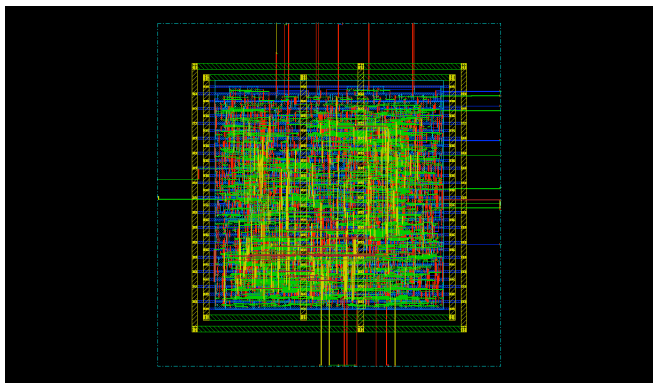

Fig. 7.2: The GDSII result for the binary logarithm realized by the Improved Parabolic Synthesis.

The floor plan is specified as 80x80μm^2. Two metals are placed for power supply and ground. The Standard Delay Format (SDF) file, which contains the timing delay information from physical design, is added with netlist to perform post layout simulation.
The SPEF file is also extracted after physical placement. It presents parasitic data information, which is used when doing post layout power estimation.
The post-layout simulation is performed to ensure the design computation correction before fabrication.

Chapter 8

Conclusion

For the Improved Parabolic Synthesis, the first subfunction and the second subfunction, $s_1(x)$ and $s_2(x)$, are developed with conformity where the desired coefficient c_1 is chosen for a value that results in both high accuracy and low complexity. An increased number of intervals in the Second-Degree Interpolation for the second subfunction, $s_2(x)$ can compensate the output precision when a simple hardware is chosen.
The truncation gives an offset to the error behavior and the optimization on the coefficients, $c_{2,i}$, $j_{2,i}$, and $l_{2,i}$, will balance it. Note that it is beneficial when developing the second subfunction, $s_2(x)$, if the difference between first help function, $f_1(x)$, and the second subfunction, s_2, is gradually decreased on the x-axis.

8.1 Comparison

Compared to the Parabolic Synthesis and CORDIC, the Improved Parabolic Synthesis is much smaller, faster, and consumes much less power.
The implementation using Improved Parabolic Synthesis and Parabolic Synthesis have an advantage of error behavior comparing to CORDIC implementation, where the error of Improved Parabolic Synthesis can be characterized to be near an normal distribution after the optimization. The Improved Parabolic Synthesis approach is suitable for a high frequency low power solution.

8.2 Future Work

The three different approaches can be implemented and prototyped on an FPGA to compare the resource utilization.
The design is possible to be implemented to be faster if it is pipelined for a smaller critical path. Alternatively, increase the number of interval in the second subfunction, $s_2(x)$, will make design faster.
The tactic of optimizing the design to characterize the error behavior should be studied and standardized.
The Improved Parabolic Synthesis can realizes other unary functions, e.g. trigonometric, exponential, square root functions, etc.

Bibliography

[1] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, vol. EC-11, no. 4, pp. 512–517, 1962.

[2] P. Tang, "Table-lookup algorithms for elementary functions and their error analysis," in *10th IEEE Symposium on Computer Arithmetic, 1991. Proceedings*, 1991, pp. 232–236.

[3] P.-T. P. Tang, "Table-driven implementation of the logarithm function in ieee floating-point arithmetic," *ACM Trans. Math. Softw.*, vol. 16, no. 4, pp. 378–400, Dec. 1990. [Online]. Available: http://doi.acm.org/10.1145/98267.98294

[4] J. Hormigo, J. Villalba, and M. Schulte, "A hardware algorithm for variable-precision logarithm," in *Application-Specific Systems, Architectures, and Processors, 2000. Proceedings. IEEE International Conference on*, 2000, pp. 215–224.

[5] J. E. Volder, "The cordic trigonometric computing technique," *Electronic Computers, IRE Transactions on*, vol. EC-8, no. 3, pp. 330–334, 1959.

[6] A. Boudabous, F. Ghozzi, M. Kharrat, and N. Masmoudi, "Implementation of hyperbolic functions using cordic algorithm," in *The 16th International Conference on Microelectronics, 2004. ICM 2004 Proceedings*, 2004, pp. 738–741.

[7] R. Andraka, "A survey of cordic algorithms for fpga based computers," in *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*. ACM, 1998, pp. 191–200.

[8] E. Hertz and P. Nilsson, "A methodology for parabolic synthesis," a book chapter in vlsi, in-tech," ISBN 978-3-902613-50-9, Tech. Rep.

[9] ——, "A methodology for parabolic synthesis of unary functions for hardware implementation," in *2nd International Conference on Signals, Circuits and Systems, SCS 2008*, 2008, pp. 1–6.

[10] ——, "Parabolic synthesis methodology implemented on the sine function," in *IEEE International Symposium on Circuits and Systems. ISCAS 2009*, 2009, pp. 253–256.

[11] P. Pouyan, E. Hertz, and P. Nilsson, "A vlsi implementation of logarithmic and exponential functions using a novel parabolic synthesis methodology compared to the cordic algorithm," in *Circuit Theory and Design (ECCTD), 2011 20th European Conference on*, 2011, pp. 709–712.

[12] Y. Voronenko and M. P. Üschel, "Multiplierless multiple constant multiplication," *ACM Transactions on Algorithms*.

[13] P. Meher, J. Valls, T.-B. Juang, K. Sridharan, and K. Maharatna, "50 years of cordic: Algorithms, architectures, and applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 9, pp. 1893–1907, 2009.

Appendix A

Logarithm Impementation Results

A.1 Area

Table A.1: ASIC synthesis area results using LPLVT

Desgin	Area(μm^2)
CORDIC	12991
Parabolic Synthesis	16368
Improved Parabolic Synthesis	4855

Table A.2: ASIC synthesis area results using LPHVT

Desgin	Area(μm^2)
CORDIC	12893
Parabolic Synthesis	16258
Improved Parabolic Synthesis	4865

Table A.3: ASIC synthesis area results using GPSVT

Desgin	Area(μm^2)
CORDIC	13061
Parabolic Synthesis	16378
Improved Parabolic Synthesis	4853

A.2 Timing

Table A.4: ASIC synthesis timing results in LPLVT

Desgin	timing path(ns)
CORDIC	86.96
Parabolic Synthesis	21.03
Improved Parabolic Synthesis	6.96

Table A.5: ASIC synthesis timing results in LPHVT

Desgin	timing path(ns)
CORDIC	not simulated
Parabolic Synthesis	38.93
Improved Parabolic Synthesis	11.90

Table A.6: ASIC synthesis timing results in GPSVT

Desgin	timing path(ns)
CORDIC	not simulated
Parabolic Synthesis	8.09
Improved Parabolic Synthesis	4.89

A.3 Power Estimation for LPHVT

Table A.7: Primetime Power Estimation for CORDIC method using LPHVT library

frequency(Hz)	Dynamic Power(nW)	Static Power(nW)	Total Power(nW)
0.001	0.0001585	43.8	44
1	0.1585	43.8	44
1.77827941004	0.281857286	43.8	44
3.16227766017	0.501221009	43.8	44
5.6234132519	0.891311	43.8	45
10	1.585	43.8	45
17.7827941004	2.818572865	43.8	47
31.6227766017	5.012210091	43.8	49
56.234132519	8.913110004	43.8	53
100	15.85	43.8	60
177.827941004	28.185728649	43.8	72
316.227766017	50.122100914	43.8	94
562.34132519	89.131100043	43.8	133
1000	158.5	43.8	202
1778.27941004	281.857286491	43.8	326
3162.27766017	501.221009137	43.8	545
5623.4132519	891.311000427	43.8	935
10000	1585	43.8	1629
17782.7941004	2818.57286491	43.8	2862
31622.7766017	5012.21009137	43.8	5056
56234.132519	8913.11000427	43.8	8957
100000	15850	43.8	15894
177827.941004	28185.7286491	43.8	28230
316227.766017	50122.1009137	43.8	50166
562341.32519	89131.1000427	43.8	89175
1000000	158500	43.8	158544
1778279.41004	281857.286491	43.8	281901
3162277.66017	501221.009137	43.8	501265
5623413.2519	891311.000427	43.8	891355
10000000	1585000	43.8	1585044
17782794.1004	2818572.86491	43.8	2818617
31622776.6017	5012210.09137	43.8	5012254

Table A.8: Primetime Power Estimation for Parabolic Synthesis method using LPHVT library

frequency(Hz)	Dynamic Power(nW)	Static Power(nW)	Total Power(nW)
0.001	6.54e-05	56.5	57
1	0.0654	56.5	57
1.77827941004	0.116299473	56.5	57
3.16227766017	0.206812959	56.5	57
5.6234132519	0.367771227	56.5	57
10	0.654	56.5	57
17.7827941004	1.162994734	56.5	58
31.6227766017	2.06812959	56.5	59
56.234132519	3.677712267	56.5	60
100	6.54	56.5	63
177.827941004	11.629947342	56.5	68
316.227766017	20.681295898	56.5	77
562.34132519	36.777122667	56.5	93
1000	65.4	56.5	122
1778.27941004	116.299473417	56.5	173
3162.27766017	206.812958975	56.5	263
5623.4132519	367.771226674	56.5	424
10000	654	56.5	711
17782.7941004	1162.99473416	56.5	1219
31622.7766017	2068.12958975	56.5	2125
56234.132519	3677.71226674	56.5	3734
100000	6540	56.5	6597
177827.941004	11629.9473417	56.5	11686
316227.766017	20681.2958975	56.5	20738
562341.32519	36777.1226674	56.5	36834
1000000	65400	56.5	65457
1778279.41004	116299.473417	56.5	116356
3162277.66017	206812.958975	56.5	206869
5623413.2519	367771.226674	56.5	367828
10000000	654000	56.5	654057
17782794.1004	1162994.73417	56.5	1163051
31622776.6017	2068129.58975	56.5	2068186

Table A.9: Primetime Power Estimation for Improved Parabolic method using LPHVT library

frequency(Hz)	Dynamic Power(nW)	Static Power(nW)	Total Power(nW)
0.001	6.12e-06	16.7	17
1	0.00612	16.7	17
1.77827941004	0.01088307	16.7	17
3.16227766017	0.0193531393	16.7	17
5.6234132519	0.0344152891	16.7	17
10	0.0612	16.7	17
17.7827941004	0.1088306999	16.7	17
31.6227766017	0.1935313928	16.7	17
56.234132519	0.344152891	16.7	17
100	0.612	16.7	17
177.827941004	1.0883069989	16.7	18
316.227766017	1.935313928	16.7	19
562.34132519	3.4415289102	16.7	20
1000	6.12	16.7	23
1778.27941004	10.8830699894	16.7	28
3162.27766017	19.3531392802	16.7	36
5623.4132519	34.4152891016	16.7	51
10000	61.2	16.7	78
17782.7941004	108.830699894	16.7	126
31622.7766017	193.531392802	16.7	210
56234.132519	344.152891017	16.7	361
100000	612	16.7	629
177827.941004	1088.30699894	16.7	1105
316227.766017	1935.31392802	16.7	1952
562341.32519	3441.52891016	16.7	3458
1000000	6120	16.7	6137
1778279.41004	10883.0699894	16.7	10900
3162277.66017	19353.1392802	16.7	19370
5623413.2519	34415.2891016	16.7	34432
10000000	61200	16.7	61217
17782794.1004	108830.699894	16.7	108847
31622776.6017	193531.392802	16.7	193548