Master's Thesis

Hardware Implementation of Number Inversion Function

Jiajun Chen
Jianan Shi

Department of Electrical and Information Technology,
Faculty of Engineering, LTH, Lund University, 2016.

Master's Thesis

Hardware Implementation of Number Inversion Function

By

Jiajun Chen and Jianan Shi

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University
SE-221 00 Lund, Sweden

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

This thesis is set in Times New Roman 11pt,
with the WORD Documentation System

© Jiajun Chen and Jianan Shi, 2015

Abstract

Nowadays, number inversion is of great significance and a complex arithmetical operator, especially the implementation in hardware. It provides a higher speed performance and lower power consumption.

Initially, the conversion of the inputs to floating-point numbers contains sign, exponent, and mantissa parts. The scope of the project is to perform an approximation of the number inversion function. In this paper, the number inversion function carried out by two methods: one based on Harmonized Parabolic Synthesis and the other one on an unrolled Newton-Raphson algorithm. It is worth mentioning that the implementation of these two methods performed as an Application Specific Integrated Circuit using a 65nm Complementary Metal Oxide Semiconductor technology with Low Power High Threshold Voltage transistors. Furthermore, the investigation and comparison for various aspects such as accuracy, error behavior, chip area, power consumption, and performance for both methods are realizable.

Keywords: harmonized parabolic synthesis, unrolled Newton-Raphson, power consumption.

Acknowledgments

This Master's thesis would not exist without the support and guidance of our primary supervisors, Erik Hertz and Peter Nilsson, who gave us considerable, useful suggestions with this thesis work and comments on the draft. We are deeply grateful of his help in the completion of this thesis.

We also owe our sincere gratitude to our supervisor, Rakesh Gangarajaiah who gave us numerous, unlimited, effective help and walked us through tough difficulties while we were stuck and confused by confronting challenging problems.

Finally, our thanks would go to our beloved family correspondingly, for their continuous support and encouragement all these years.

Peter Nilsson
in memoriam

Peter Nilsson is our supervisor during the whole project. He gave us tremendous help when we confronted challenging problems and left us comments to the draft before he passed away. We feel great sorrow that he is absent and unable to share the achievement of the completion of this thesis.

Jiajun Chen
Jianan Shi

Contents

List of Acronyms

CORDIC	COordinated Rotation DIgital Computer
CMOS	Complementary Metal Oxide Semiconductor
dB	Decibel
GPSVT	General Purpose Standard Threshold Voltage
HPS	Harmonized Parabolic Synthesis
LPHVT	Low Power High Threshold Voltage
LPLVT	Low Power Low Threshold Voltage
LUT	Look-UpTable
NR	Newton-Raphson algorithm
P&R	Place and Route
RMS	Root Mean Square
VHDL	Very high speed integrated circuit Hardware Description Language

List of Figures

List of Tables

CHAPTER 1

1 Introduction

Unary function, especially non-linear operations such as sine, logarithmic, exponential and number inversion functions have been widely applied in computer graphics, wireless systems and digital signal processing [1]. These functions are simple in software for low-precision cases. However, concerning high precision and high-speed applications, software implementations are insufficient. Therefore, hardware implementations for these functions are worth considering.

To implement these non-linear operations in hardware, there are several different methods to consider, i.e. Look-Up Table (LUT), COordinated Rotation DIgital Computer (CORDIC) algorithm, Harmonized Parabolic Synthesis (HPS) and unrolled Newton-Raphson (NR) algorithm. In this thesis, LUT and CORDIC are briefly introduced. On the contrary, we implement HPS and NR method both in software and in hardware. Furthermore, the investigation and comparison, which are of great interest, for various factors such as accuracy, error behavior, chip area, power consumption, and performance.

For accuracies up to about 12 bit, a LUT is simple, fast and strait forward method to implement approximations in hardware. Reading a value from the table or matrix is much faster than calculating the number with an algorithm or using a trigonometric function. However, the primary drawback of a LUT is its memory usage. In many cases, when higher precision is required, the size of the LUT increases exponentially, which is not always feasible.

A CORDIC algorithm is a simple and efficient method to calculate hyperbolic and trigonometric functions. The basic idea is vector rotation with a number of stages to improve accuracy. Adders/subtractors are required to determine the direction of each rotation. The primary advantage is that hardware multiplications are avoidable, which decreases the complexity of the design. Based on the CORDIC algorithm, only simple shift-and-add operations are required for tasks such as real and complex multiplications, divisions, square root calculations, etc. However, the latency of the implementation is an inherent drawback of the conventional CORDIC algorithm [2]. Furthermore, it requires n+1 iterations to obtain n-bit precision, which leads to low speed carry-propagate additions, low throughput rate and chip area consuming shift operations.

Erik Hertz and Peter Nilsson [3] have recently proposed a parabolic synthesis method. It is an approximation method for the implementations of unary functions. It is a new method with high-speed performance. It contains pre-processing,

processing, and post-processing part. By combining with a synthesis of parabolic functions, the approximation of the target function, such as sine, exponential, logarithm and number inversion functions, is realizable. The improved method based on parabolic synthesis, HPS, requires only two sub-functions, combined with second-order interpolation, which results in better speed performance.

The NR iteration algorithm is also applicable. It is a method named after Isaac Newton and Joseph Raphson to find approximations to the roots of a real-valued function. This method is hardware-friendly. In this algorithm, A LUT is often required to select a start value, also called as an initial guess. It is important for the initial guess to be as close as possible to the true result. To meet the bit precision requirement, several iterations are necessary. It is worth mentioning that more iterations yield smaller sizes of the LUTs. Therefore, there is a trade-off between the number of iterations and the size of LUTs.

In this thesis, we focus on the HPS and NR method. There are totally five architectures implemented in hardware based on HPS and NR method. The comparison of the hardware design concerning various aspects such as accuracy, error behavior, chip area, power consumption and speed performance achieved. The power consumption, both static and dynamic, estimated for different clock frequencies. When applying HPS methods, different numbers of intervals in the second sub-function have a large influence on the error behavior, critical path, and chip area. Hence, three different architectures built using 16, 32, and 64 intervals in the second sub-function. Two different Matlab models designed based on both HPS and NR methods. All implementations designed in Very High Speed Integrated Circuit Hardware Description Language (VHDL) and synthesized using a 65nm Complementary Metal Oxide Semiconductor (CMOS) technology. Various synthesis libraries such as Low Power High Threshold Voltage (LPHVT), Low Power Low Threshold Voltage (LPLVT), and General Purpose Standard Threshold Voltage (GPSVT) are considered.

As a result, chip area, timing and power consumption using different clock frequencies reported. Furthermore, the physical layouts for both HPS and NR method after Place and Route (P&R), demonstrated as well.

1.1 Thesis Outlines

Remaining chapters, listed as follows:

Chapter 2 introduces general algorithm for number inversion function.

Chapter 3 describes the detailed Parabolic Synthesis theory.

Chapter 4 demonstrates HPS theory.

Chapter 5 illustrates NR theory.

Chapter 6 describes general hardware architecture using HPS method.

Chapter 7 explains general hardware architecture using NR method.

Chapter 8 expounds error behavior analysis in Matlab.

Chapter 9 introduces three different implementations regarding to three different intervals by using HPS and two different implementations with respect to two different number of NR iterations. All five detailed architectures are included.

Chapter 10 lists and analyzes the result of Chapter 9, including error behavior, error metrics, chip area, timing and power consumption.

Chapter 11 contains conclusion and future work of this thesis work.

Appendix A lists the coefficients in the second sub-function using HPS method, the coefficients in LUTs of NR iteration method, and the power consumptions at various frequencies for all architectures.

CHAPTER 2

2 Algorithm for number inversion

In this thesis, the algorithms for number inversion function are applicable by using floating-point numbers [4], containing a mantissa and an exponent. It is worth mentioning that the exponent scales the mantissa [5]. The separation for the computation for both the mantissa and the exponent is achievable. This separation results in the reduction of the computation complexity due to the approximation is on a limited mantissa range. In addition, the computation of the exponent is simple in hardware. Table 2.1 describes the transformation of a fixed-point number to a floating-point number in number base 2.

Table 2.1 The transformation of a fixed-point number to a floating-point number in number base 2.

Base 10	158															
Fixed-point base 2	0000000010011110															
Exponent	0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	0
Index	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Floating-point base 2	$1.001111000000000 \cdot 2^7$															

The exponent is depending on the most significant bit of the fixed-point number. The exponent when using floating-point representation thus scales the mantissa. The implementation is computing the mantissa of binary function as an approximation. Figure 2.1 demonstrates the block diagram of the approximation.

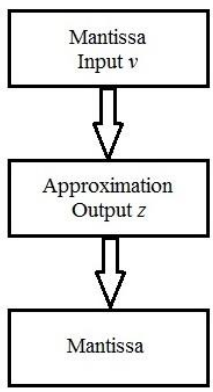

Fig 2.1 The block diagram of the approximation.

As shown in Figure 2.1, the input *v* is the mantissa part of the floating-point number. The output *z* is the output after the approximation.

2.1 Number inversion function

As mentioned in Table 2.1, the transformation of a fixed-point number to a floating-point number is realizable. When computing the number inversion function, the output *z*, shown in (2.2).

$$z = \frac{1}{V_0 V_{-1} V_{-2} \ldots V_{-15} \cdot 2^{index}} \tag{2.2}$$

where V_0 is the integer bit of *v*, v_{-1}, v_{-2}, …, v_{-15} are the fractional bits of *v*.

In (2.3) is (2.2) modified.

$$z = \frac{1}{V_0 V_{-1} V_{-2} \ldots V_{-15}} \cdot 2^{-index} \tag{2.3}$$

which explains why the index is negative.

Figure 2.2 illustrates the inverse function in the range from 1 to 2.

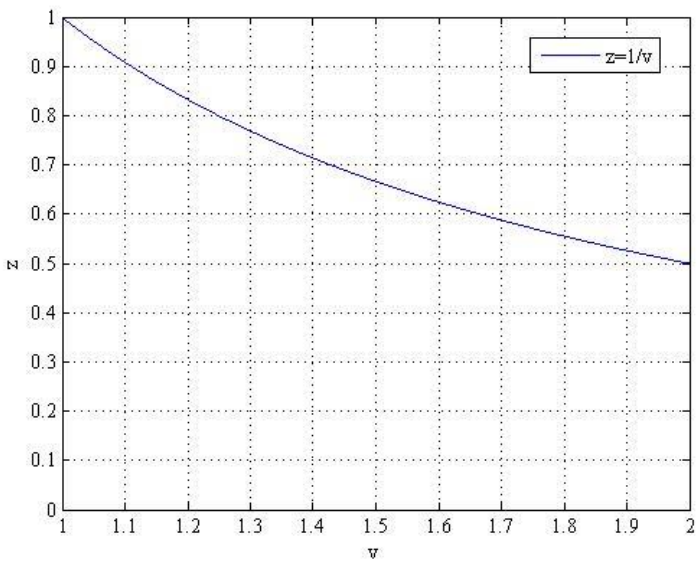

Fig 2.2 The curve of number inversion function.

The algorithm of the number inversion function, illustrated in Figure 2.3.

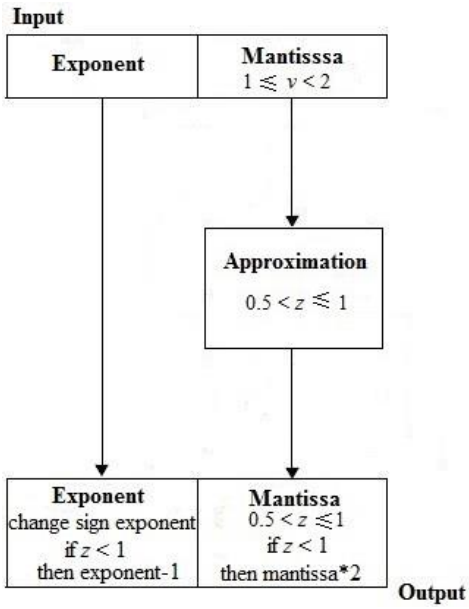

Fig 2.3 The block diagram of number inversion.

The input is a floating-point number with an exponent part and a mantissa. As shown in Figure 2.3, the computation for the mantissa and the exponent is separate. For the exponent, the sign bit changes depending on the result. Concerning the mantissa, if the mantissa of the result is less than 1, the exponent has to be subtracted with 1. In addition, if the mantissa is less than 1, multiply the mantissa with 2. Note that, in this thesis, we focus on the approximation part, performed by HPS and NR method, which are in further chapters.

CHAPTER 3

3 Parabolic Synthesis

This chapter contains the description of the Parabolic Synthesis algorithm. It is an approximation method for implementing unary functions. Parabolic Synthesis is the product of a series of second order sub-functions [6], defined as $s_1(x)$, $s_2(x)$… $s_n(x)$ to approximate the original function, defined as $f_{org}(x)$, shown in (3.1).

$$f_{org}(x) = s_1(x) \cdot s_2(x) \cdot s_3(x) \cdot \ldots \cdot s_n(x) \tag{3.1}$$

The number of sub-functions affects the accuracy. When the number of sub-functions, n, goes to infinite, the original function, $f_{org}(x)$, is fully satisfied. In order to develop these sub-functions, the help function is used. The first help function, $f_1(x)$, is defined as the quotient of the original function, $f_{org}(x)$, and the first sub-function, $s_1(x)$, as shown in (3.2).

$$f_1(x) = \frac{f_{org}(x)}{s_1(x)} = s_2(x) \cdot s_3(x) \cdot \ldots \cdot s_\infty(x) \tag{3.2}$$

In the same manner, the following help functions, $f_n(x)$, is defined as shown in (3.3). Note that, the n^{th} help function is to develop the n^{th}+1 sub-function.

$$f_n(x) = \frac{f_{n-1}(x)}{s_n(x)} = s_{n+1}(x) \cdot s_{n+2}(x) \cdot \ldots \cdot s_\infty(x) \tag{3.3}$$

3.1 First sub-function

In order to apply Parabolic Synthesis method, the normalization of the target function is essential, to the range $0 \le x < 1$ on the x-axis and $0 \le y < 1$ on the y-axis. This normalization results in the original function, $f_{org}(x)$.

Figure 3.1 demonstrates an example of an original function, $f_{org}(x)$.

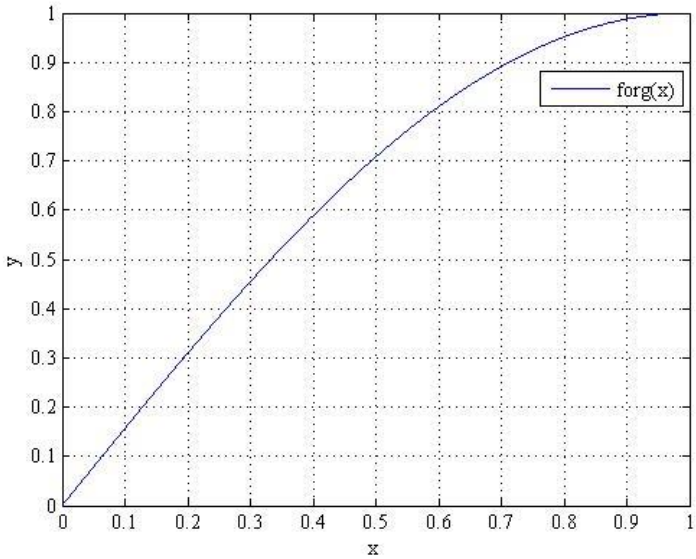

Fig 3.1 An example of an original function, $f_{org}(x)$.

In addition, the original function, $f_{org}(x)$, should satisfy the requirements as follows:

1. The original function, $f_{org}(x)$, should be strict concave or convex. This is due to that the sub-functions are second order polynomials which are strict concave or convex.
2. The first help function, $f_1(x)$, has a limited value when x goes to 0. This indicates that if the original function, $f_{org}(x)$, does not have a limited value when x goes to 0, the first help function, $f_1(x)$, is not defined at all.
3. The limited value in requirement 2 should be smaller than 1 or larger than -1 after it is subtracted with 1. If the limited value in requirement 2 is not inside the interval, the first sub-function, $s_1(x)$, is not deployable since the gradient is not positive in the interval.

The first sub-function, $s_1(x)$, is a second order polynomial, as shown in (3.4).

$$s_1(x) = l_1 + k_1 x + c_1(x - x^2) \tag{3.4}$$

As shown in Figure 3.1, the first sub-function, $s_1(x)$, should cut the same starting point and ending point as the original function, $f_{org}(x)$, since it is to approach $f_{org}(x)$. The starting point is (0, 0) which gives l_1 to be 0. The coefficient k_1, calculated to be 1, based on both the starting point and the ending point. The first sub-function, $s_1(x)$, is thus simplified according to (3.5).

$$s_1(x) = x + c_1(x - x^2) \tag{3.5}$$

The coefficient c_1, calculated according to (3.6).

$$1 = \lim_{x \to 0} \frac{f_{org}(x)}{s_1(x)} = \lim_{x \to 0} \frac{f_{org}(x)}{x + c_1(x - x^2)} \tag{3.6}$$

Note that, since x^2 goes faster towards 0 than x, the limit in (3.6), modified as shown in (3.7).

$$1 = \lim_{x \to 0} \frac{f_{org}(x)}{(1 + c_1)x} \tag{3.7}$$

The coefficient c_1, calculated according to (3.8).

$$c_1 = \lim_{x \to 0} \frac{f_{org}(x)}{x} - 1 \tag{3.8}$$

3.2 Second sub-function

The second sub-function, $s_2(x)$, is designed to approach the first help function, $f_1(x)$. Figure 3.2 demonstrates an example of the first help function, $f_1(x)$.

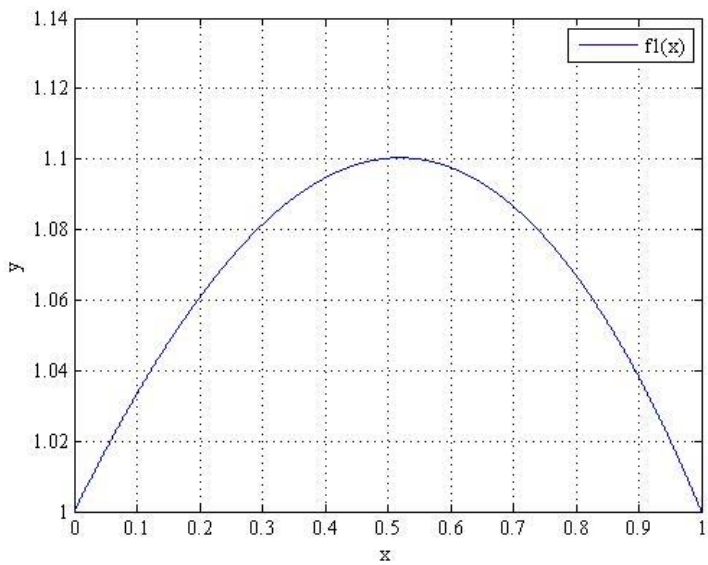

Fig 3.2 An example of a first help function, $f_1(x)$.

The second sub-function, $s_2(x)$, is a second order polynomial as well, defined in (3.9).

$$s_2(x) = 1_2 + k_2 x + c_2(x - x^2) \tag{3.9}$$

Figure 3.3 demonstrates an example of a comparison between a first help function, $f_1(x)$, and a second sub-function, $s_2(x)$.

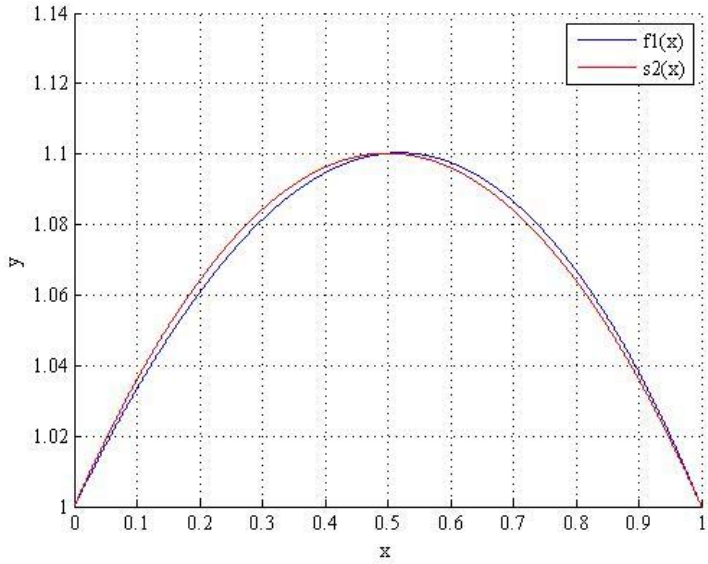

Fig 3.3 A second sub-function, $s_2(x)$, compared to a first help function, $f_1(x)$.

Since the second sub-function, $s_2(x)$, is to approach the first help function, $f_1(x)$, both functions should cut the same starting point (0, 1), and the same ending point (1, 1). The coefficient l_2, calculated to be 1, based on the starting point, and the coefficient k_2, calculated to be 0, based on both the starting point and ending point. The second sub-function, $s_2(x)$, is simplified as shown in (3.10).

$$s_2(x) = 1 + c_2(x - x^2) \tag{3.10}$$

The coefficient c_2, is to ensure the quotient of $f_1(x)$ and $s_2(x)$ to be 1 when x equals 0.5. In (3.11), is the coefficient, c_2, calculated.

$$c_2 = 4(f_1(\frac{1}{2}) - 1) \tag{3.11}$$

Similarly, by dividing the first help function, $f_1(x)$, with the second sub-function, $s_2(x)$, an example of a second help function, $f_2(x)$, is illustrated in Figure 3.4.

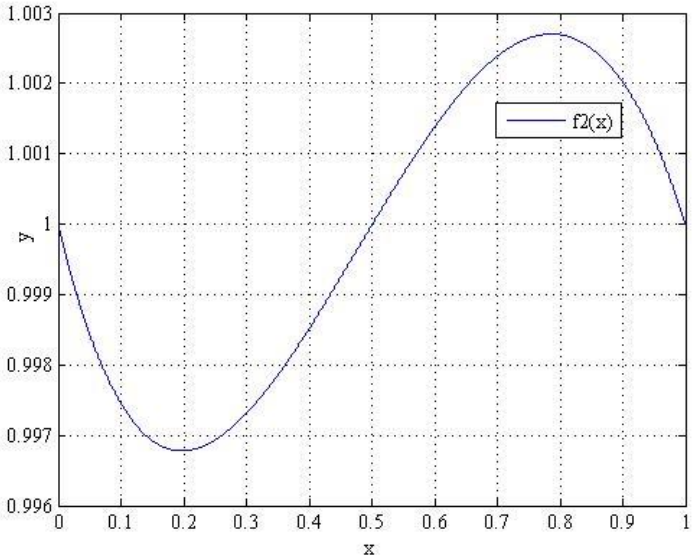

Fig 3.4 An example of a second help function, $f_2(x)$.

As shown in Figure 3.4, the second help function, $f_2(x)$, contains both convex and concave curves. To apply this methodology, the second help function, $f_2(x)$, should be segmented into two intervals and both intervals should be renormalized.

3.3 N^{th} sub-function when $n>2$

When developing the third help function, $s_3(x)$, the second help function, $f_2(x)$, should be renormalized into two intervals due to requirement 1, mentioned in Section 3.1. As shown in Figure 3.4, the second help function appears both convex and concave curves. By dividing the interval into two intervals, $0 \le x < 0.5$ and $0.5 \le x < 1$, the curve is strict convex or concave again.

Hence, the second help function, $f_2(x)$, is shown in (3.12).

$$f_2(x) = \begin{cases} f_{2,0}(x) & 0 \le x < 0.5 \\ f_{2,1}(x) & 0.5 \le x < 1 \end{cases} \tag{3.12}$$

The second help function, $f_2(x)$, consists of a pair of concave and convex curves. The renormalization for both these curves is essential, to the interval $0 \le x < 1$ on the x-axis.

Figure 3.5 demonstrates partially the segmented and renormalized second help function, $f_2(x)$.

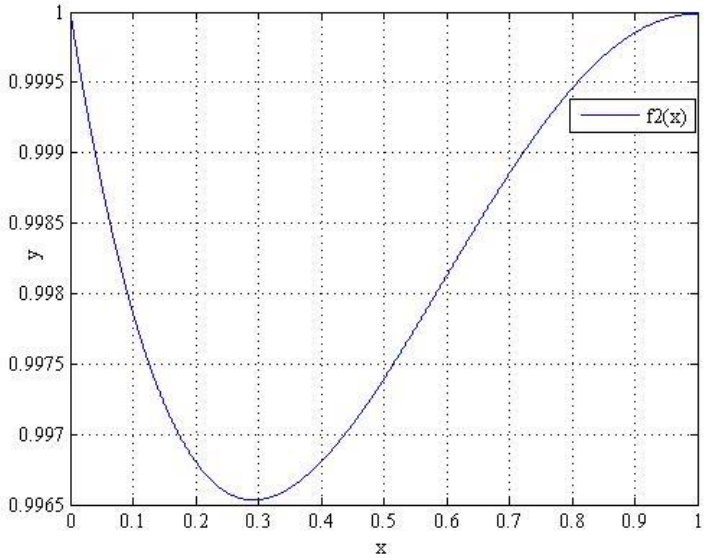

Fig 3.5 The segmented and renormalized second help function, $f_2(x)$.

It is worth mentioning that x_3 replaces x based on (3.14). The purpose of this replacement is to map the input x to the normalized parabolic curve.

$$x_3 = fract(2x) \tag{3.13}$$

The third sub-function, $s_3(x)$, for each interval, is shown in (3.14).

$$s_3(x) = \begin{cases} s_{3,0}(x) & 0 \le x < 0.5 \\ s_{3,1}(x) & 0.5 \le x < 1 \end{cases} \tag{3.14}$$

In the same manner, the n^{th} help function, $f_n(x)$, can be developed according to (3.15). Along with the growth of n, the number of pairs of concave and convex curves is higher.

$$f_n(x) = \begin{cases} f_{n,0}(x) & 0 \le x < \dfrac{1}{2^{n-2}} \\ f_{n,1}(x) & \dfrac{1}{2^{n-2}} \le x < \dfrac{2}{2^{n-2}} \\ \cdots & \cdots \\ \cdots & \cdots \\ f_{n,2^{n-1}-1}(x) & \dfrac{2^{n-2}-1}{2^{n-2}} \le x < 1 \end{cases} \tag{3.15}$$

Hence, the n^{th} sub-function, $s_n(x)$, can be developed as shown in (3.16).

$$s_n(x) = \begin{cases} s_{n,0}(x_n) & 0 \le x < \dfrac{1}{2^{n-2}} \\ s_{n,1}(x_n) & \dfrac{1}{2^{n-2}} \le x < \dfrac{2}{2^{n-2}} \\ \cdots & \cdots \\ \cdots & \cdots \\ s_{n,2^{n-2}-1}(x_n) & \dfrac{2^{n-2}-1}{2^{n-2}} \le x < 1 \end{cases} \tag{3.16}$$

It is worth mentioning that x_n replaces x, as shown in (3.17). Similar to the replacement in the third sub-function, $s_3(x)$, this replacement is to map the input x to the normalized parabolic curves.

$$x_n = fract(2^{n-2}x) \tag{3.17}$$

The n^{th} sub-function, $s_n(x)$, thus partially described as $s_{n,m}(x_n)$, shown in (3.18).

$$s_{n,m}(x_n) = 1 + c_{n,m}(x_n - x_n^2) \tag{3.18}$$

The coefficient, $c_{n,m}$, can be calculated in the same manner as in (3.11), shown in (3.19).

$$c_{n,m} = 4(f_{n-1,m}(\frac{2(m+1)-1}{2^{n-1}}) - 1) \tag{3.19}$$

CHAPTER 4

4 Harmonized Parabolic Synthesis

Based on the Parabolic Synthesis, a superior methodology, called Harmonized Parabolic Synthesis (HPS), only requires two sub-functions, as described in (4.1). Both the sub-functions are second order polynomials. Note that, the second sub-function, $s_2(x)$, is combined with second-degree interpolation. The output accuracy thus depends on the number of intervals in the interpolation.

$$f_{org}(x) = s_1(x) \cdot s_2(x) \tag{4.1}$$

Similar to the Parabolic Synthesis, the normalization of the original function, $f_{org}(x)$, is essential, to the range $0 \leq x < 1$ on the x-axis and $0 \leq y < 1$ on the y-axis. Additional requirements for the original function, $f_{org}(x)$, listed as follows:

1. The original function, $f_{org}(x)$, should be strict concave or convex. This is due to that the sub-functions are second order polynomials which are strict concave or convex.
2. The first help function, $f_1(x)$, has a limited value when x goes to 0. This indicates that if the original function, $f_{org}(x)$, does not have a limited value when x goes to 0, the first help function, $f_1(x)$, is not defined at all.

4.1 First sub-function

The first sub-function, $s_1(x)$, is determined to approach the original function, $f_{org}(x)$, as defined in (4.2).

$$s_1(x) = l_1 + k_1 x + c_1(x - x^2) \tag{4.2}$$

Since the first sub-function, $s_1(x)$, has the same starting point and the ending point as the original function, $f_{org}(x)$, the coefficient l_1, calculated based on the starting point, and the coefficient k_1, calculated based on both the starting point and ending point. The coefficient, c_1, is selected by scanning from the interval (-1.5, 0) and (0, 1.5) respectively based on the concavity and convexity of the original function, $f_{org}(x)$.

4.2 Second Sub-function

The second sub-function, $s_2(x)$, is determined to approach the first help function, $f_1(x)$, as explained in Section 3.2. The former first help function, $f_1(x)$, is divided into 4 intervals as shown in Figure 4.1.

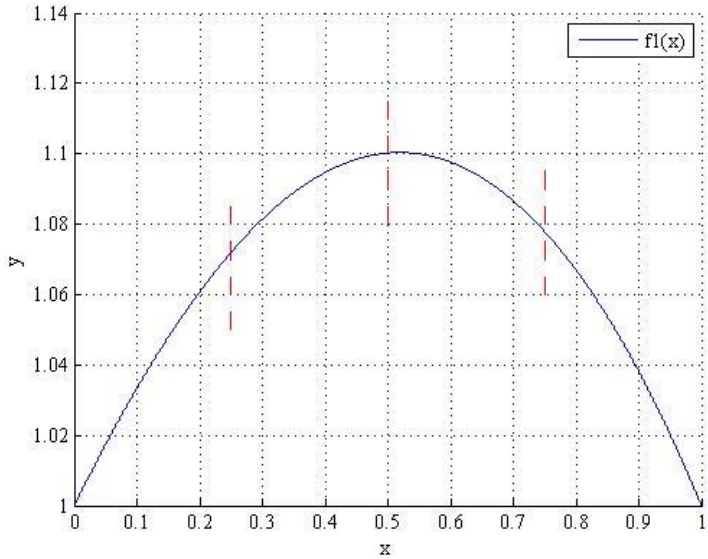

Fig 4.1 The first help function, divided evenly.

The number of intervals is 2^n. Along with the growth of n, the improvement of the precision is realizable. Concerning the second sub-function, $s_2(x)$, with respect to the i^{th} interval, the general formula can be expounded according to (4.3).

$$s_{2,i}(x) = l_{2,i} + k_{2,i}x_\varphi + c_{2,i}(x_\varphi - x_\varphi^2) \tag{4.3}$$

where i is from 0 to $2^{n-1} - 1$.

The first help function in each interval, defined as $f_{1,i}(x)$, is approached by the second sub-function, $s_{2,i}(x)$, in each corresponding interval. To calculate all the coefficients, three specific coordinates in each interval, which are the starting point, the middle point, and the ending point, are important. The general formula of the second sub-function, $s_{2,i}(x)$, in each interval can be expounded according to (4.4).

$$s_2(x) = \begin{cases} s_{2,0}(x_\varphi) & 0 \le x < \dfrac{1}{2^\varphi} \\ s_{2,1}(x_\varphi) & \dfrac{1}{2^\varphi} \le x < \dfrac{2}{2^\varphi} \\ \cdots & \cdots \\ \cdots & \cdots \\ s_{2,i-1}(x_\varphi) & \dfrac{i-1}{2^\varphi} \le x < 1 \end{cases} \tag{4.4}$$

It is worth mentioning that x_φ replaces x. The purpose of this replacement is to maintain the normality in each interval by multiplying by 2^φ, according to (4.5).

$$x_\varphi = fract(2^\varphi x) \tag{4.5}$$

In order to compute the coefficient, $l_{2,i}$, in each interval, the starting point of each interval is essential, as shown in (4.6).

$$l_{2,i} = f_1(start, i) \tag{4.6}$$

When computing the coefficient, $k_{2,i}$, which is the slope in each interval, is expressed according to (4.7).

$$k_{2,i} = f_1(start, i) - f_1(end, i) \tag{4.7}$$

Computing the coefficient, $c_{2,i}$, requires the help of the middle point in each interval, as shown in (4.8).

$$c_{2,i} = 4(f_{1,i}(\frac{1}{2}) - l_{2,i} - \frac{1}{2} k_{2,i}) \tag{4.8}$$

The second sub-function in each interval, $s_{2,i}(x)$, can be modified based on (4.3), as shown in (4.9).

$$s_{2,i}(x) = l_{2,i} + (k_{2,i} + c_{2,i})x_\varphi - c_{2,i}x_\varphi^2 \tag{4.9}$$

This modification saves an adder in hardware by adding $k_{2,i}$ and $c_{2,i}$ manually, defined as $j_{2,i}$, according to (4.10).

$$j_{2,i} = k_{2,i} + c_{2,i} \tag{4.10}$$

According to (4.9) and (4.10), the second sub-function in each interval is according to (4.11).

$$s_{2,i}(x) = l_{2,i} + j_{2,i}x_\varphi - c_{2,i}x_\varphi^2 \tag{4.11}$$

4.3 Selection of a proper c_1

The selection of the coefficient, c_1, is extremely essential since c_1 is not only to meet the precision requirement of the output, but also to realize a hardware friendly architecture. The steps of developing c_1 combined with the number of intervals in the second sub-function, $s_2(x)$, listed as follows:

First, select every value from the interval (-1.5, 0) and (0, 1.5) respectively based on the convexity and concavity of the original function, $f_{org}(x)$. Depending on the different values of c_1, the first sub-function, $s_1(x)$, is settled. Second, the

second sub-function, $s_2(x)$, is developed based on (4.1). Finally, the computation of the precision of the output is realizable with these two sub-functions.

The error $e(x)$ described according to (4.12).

$$e(x) = abs(f_{org}(x) - s_1(x) \cdot s_2(x)) \tag{4.12}$$

When presenting the accuracy of a function, the method of using logarithms provides a great resolution. We propose the concept of Decibel (dB) [8].

4.4 Bit Precision

4.4.1 Definition

There are two equal ways of defining the result in decibels. With respect to power measurements, described according to (4.13).

$$P_{dB} = 10 \log_{10}(\frac{p}{p_0}) \tag{4.13}$$

where p is the measured power and p_0 the reference power. With respect to the measurements of amplitude, described according to (4.14).

$$A_{dB} = 20 \log_{10}(\frac{a}{a_0}) \tag{4.14}$$

where a is the measured amplitude and a_0 the reference amplitude.

4.4.2 Combination of Decibel and Binary Number

This combination provides a great result. In Binary representation, the transformation of the numbers in number base 2 to decibel is shown in (4.15).

$$20 \log_{10} 2 \approx 6dB \tag{4.15}$$

Since 6dB represents 1 bit in resolution, this transformation leads to a simplified comprehension of the result. For instance, if the error equals 0.001, the transformation, described according to (4.16).

$$20 \log_{10} 0.001 = -60dB \tag{4.16}$$

which represents 10 bits in resolution.

4.5 An example of c_1 selection

As a result, the bit precision is a function of c_1 combined with the number of intervals in the second sub-function, $s_2(x)$. Figure 4.2 [9] demonstrates an example of c_1 selection.

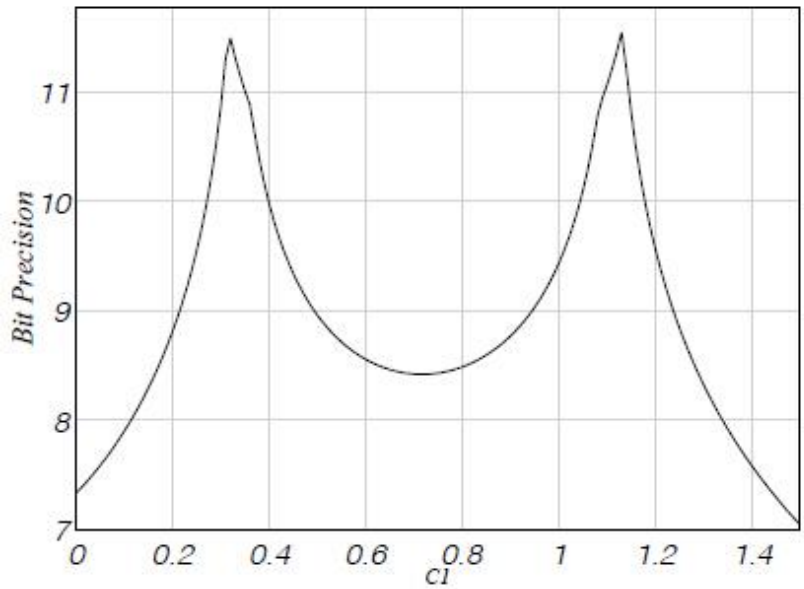

Fig 4.2 C_1 selection with 1 interval in the second sub-function, $s_2(x)$.

There are two peak values approximately 0.3 and 1.1 respectively. The precision can be greater than 11 bits. However, to realize a hardware friendly architecture, these peak values are not always feasible. A simple hardware architecture is realized by selecting $c_1$as 0, 0.5, or 1. Note that if the graph of the original function is convex, the sweeping range of c_1 is from -1.5 to 0. Another method to increase the freedom of selecting c_1 is to increase the number of intervals in the second sub-function, $s_2(x)$.

CHAPTER 5

5 Newton-Raphson

In general, the NR methodology [10] is an iterative process to approximate the root of a function. The NR iteration, as described in (5.1).

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{5.1}$$

In (4.1), x_n is from the previous iteration, $f(x_n)$ stands for the value at x_n. The derivative of $f(x_n)$, $f'(x_n)$ stands for the slope of $f(x_n)$ at x_n. The number n is the number of iteration subtracted with 1.

Hence, the first iteration, as expounded in (5.2).

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{5.2}$$

where x_0 is the initial guess for a single root σ of the equation.

5.1 An example of NR iterations

For instance, assume that the equation, $f(x)$, illustrated in (5.3), is to be approximated. The precision requirement is that the error is less than 0.000000001.

$$f(x) = 3x^2 - 6x + 2 \tag{5.3}$$

Figure 5.1 demonstrates the graph of the equation, $f(x)$.

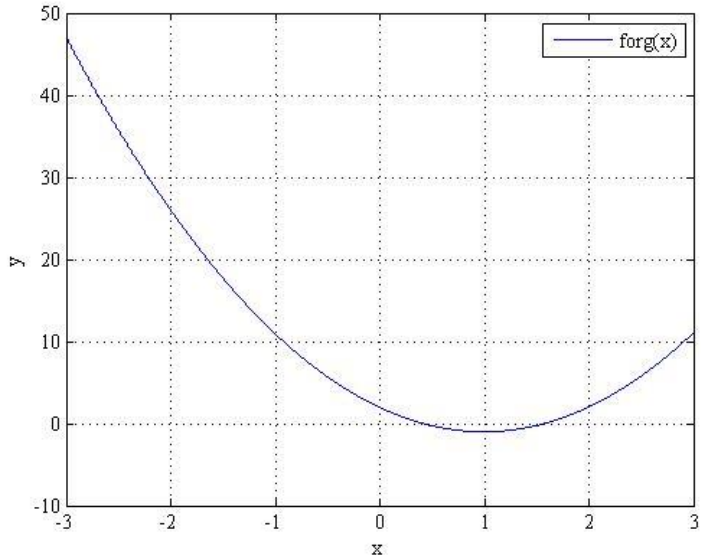

Fig 5.1 The curve of the equation, *f*(*x*).

The derivative of *f*(*x*), $f'(x)$ is shown in (5.4).

$$f'(x) = 6x - 6 \tag{5.4}$$

There are two roots in the interval (0, 1) and (1, 2) respectively. To calculate the root in the interval (0, 1), assume that the initial guess x_0 equals 0.5. Assume that 15 decimal places are reserved. The first iteration, is as shown in (5.5).

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 0.416666666666667 \tag{5.5}$$

The second iteration is as described in (5.6).

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 0.422619047619048 \tag{5.6}$$

Furthermore, the third iteration is as shown in (5.7).

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = 0.422649729995091 \tag{5.7}$$

The true value of the root σ in the interval (0, 1) is approximately 0.422649730810374. Therefore, the error *e* is as shown in (5.8).

$$e = abs(x_3 - \sigma) = 0.000000000815283 < 0.000000001 \tag{5.8}$$

After applying only three iterations, the precision requirement is fully satisfied. Hence, it is critical for x_0 to be as close as possible to the real root. Note that if the equation, $f(x)$, is a continuous derivative, the iterations will strictly converge to σ by developing an initial guess x_0 that is close enough to σ.

In the example above, to meet the precision requirement, three iterations are required. However, this may not be applicable for most of the cases nowadays. In 1991, Paolo Montuschi and Marco Mezzalama discussed the possibility of using absolute error instead of relative error in order to apply fixed number of iterations, specifically, only one or two iterations [11]. That leads to an optimization to minimize the absolute error with a good initial guess. In 1994, Michael J. Schulte, Earl E. Swartzlander and J. Omar discussed how to optimize the initial guess for reciprocal functions [12]. They focused on the optimization of the initial guess interval by interval to minimize the absolute error. The general formula is as illustrated in (5.9).

$$\beta_n = \frac{p^{2-n} + q^{2-n}}{p^{2-n}q + q^{2-n}p} \tag{5.9}$$

where β_n stands for the initial guess, n is the number of iteration, p and q are the circumscription of the interval.

This formula can be realizable by applying LUTs to select a proper initial guess β_n that is close enough to the true value σ, which leads to an increased speed of the NR convergence.

5.2 Reciprocal, square root, square root reciprocal

The NR methodology provides tremendous unary function approximations such as reciprocal (number inversion), square root, and square root reciprocal by using different setups of the initial guess, x_0, described as follows:

If $f(x)$ is set to (5.10),

$$f(x) = \frac{1}{x} - a \tag{5.10}$$

The modification of the general iteration in (5.1) is according to (5.11).

$$x_{n+1} = x_n(2 - ax_n) \tag{5.11}$$

These iterations go to $1/a$ so that the reciprocals can be calculated.

If $f(x)$ is set to (5.12),

$$f(x) = x^2 - a \tag{5.12}$$

The modification of the general iteration in (5.1) is according to (5.13).

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right) \tag{5.13}$$

These iterations go to $\sqrt{a}$ so that the square root can be realizable.

If $f(x)$ is set to (5.14),

$$f(x) = \frac{1}{x^2} - a \tag{5.14}$$

The modification of the general iteration in (5.1) is according to (5.15).

$$x_{n+1} = \frac{x_n}{2}\left(3 - ax_n^2\right) \tag{5.15}$$

These iterations go to $1/\sqrt{a}$ so that the square root reciprocals can be calculated. It is worth mentioning that this setup can also be used to calculate $\sqrt{a}$ by simply multiplying it by a.

CHAPTER 6

6 Hardware Architecture Using HPS

The HPS procedure consists of three steps: pre-processing, processing, and post-processing, as shown in Figure. 6.1.

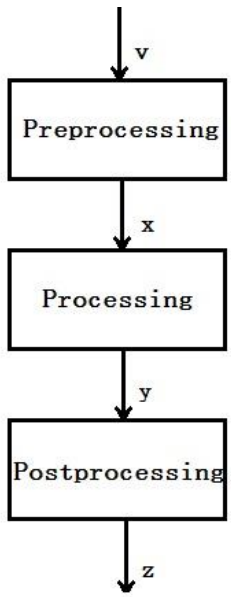

Fig 6.1 The procedure of HPS.

Both the pre-processing part and the post-processing part are conversion steps. The processing part is the approximation step.

6.1 Pre-processing

In order to ensure that the output *x* is in the interval (0, 1), the conversion of the input *v* is essential. An example of sine function, described in (6.1).

$$f(v) = \sin(v) \tag{6.1}$$

The input domain of *v* is from 0 to π/2. In order to normalize the output *x* to be in the interval (0, 1), the input *v* should be multiplied by 2/π. Hence, the original function is as shown in (6.2).

$$f_{org}(x) = \sin(\frac{\pi}{2} x) \tag{6.2}$$

6.2 Processing

6.2.1 Architecture of HPS

According to (4.1), (4.2), and (4.11), the architecture of HPS can be demonstrated according to Figure 6.2.

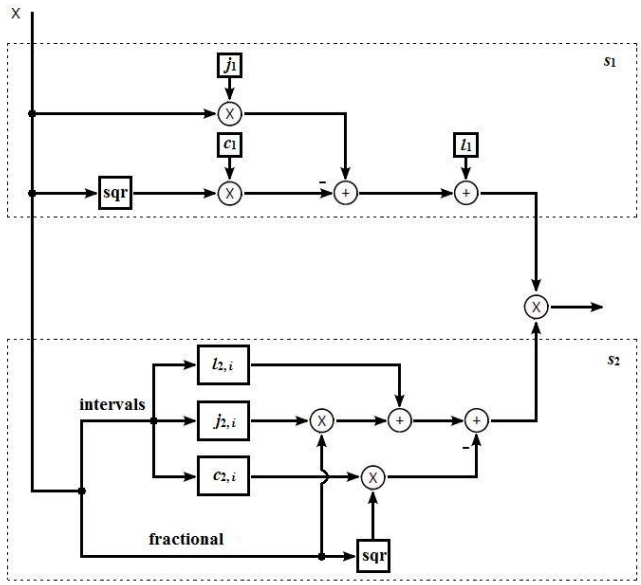

Fig 6.2 The architecture of the processing step using HPS with 16 intervals.

With respect to the first sub-function, $s_1(x)$, and the second sub-function, $s_2(x)$, a squarer unit should be applied. It is worth mentioning that, if the coefficient, c_1, is selected to 0, the squarer unit in the first sub-function, $s_1(x)$, is no longer required.

6.2.2 Squarer

As mentioned in Section 6.2.1, a squarer unit is required. Figure 6.3 illustrates the squarer unit.

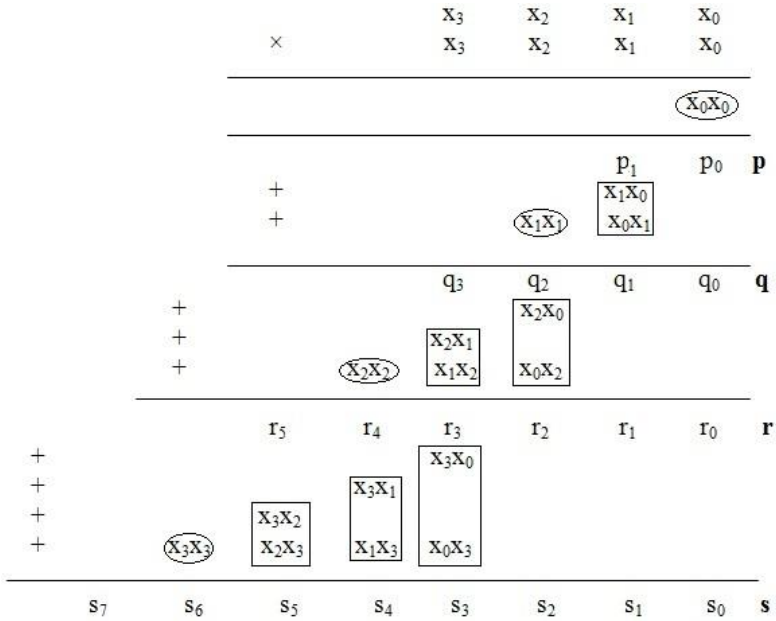

Fig 6.3 The squarer unit algorithm.

As shown in Figure 6.3, the computation of the output, ***p***, is realizable by squaring the least significant bit of *x*. The computation of the output, ***q***, is also realizable by squaring the two least significant bits of *x*. The computation of the remaining ***r*** and ***s*** can be realizable in the same manner. By applying this algorithm, the implementation of the partial output of x_n^2 is realizable, which leads to a reduction of chip area compared to the multiplier counterpart.

However, there are plenty of adders in the algorithm shown in Figure 6.3. To save adders, Erik Hertz [13] put an optimized algorithm forward. The reduction of the ellipses containing x_0x_0, x_1x_1, x_2x_2, and x_3x_3 is realizable, to x_0, x_1, x_2, and x_3 respectively. Moreover, it is realizable to move all the squares to the next column. The modification of the algorithm in Figure 6.3 is as shown in Figure 6.4.

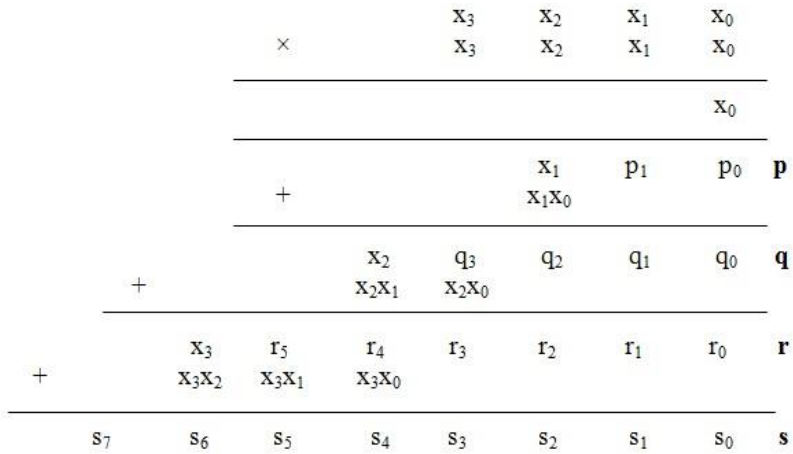

Fig 6.4 The optimized squarer unit.

The factor, p_0, in the partial product, ***p***, can be optimized according to (6.4).

$$p_0 = x_0 x_0 = x_0 \tag{6.4}$$

Since p_0 cannot have an influence to p_1, p_1 equals 0. The value of q_1 is as shown in (6.5).

$$q_1 = p_1 \cdot 2^1 + x_1 x_0 \cdot 2^1 + x_0 x_1 \cdot 2^1 = 0 \cdot 2^1 + x_1 x_0 \cdot 2^2 = 0 \cdot 2^1 \tag{6.5}$$

The value of q_2 is as illustrated in (6.6).

$$q_2 = x_1 x_0 \cdot 2^2 + x_1 x_1 \cdot 2^2 = x_1 \cdot 2^2 + x_1 x_0 \cdot 2^2 \tag{6.6}$$

The value of r_2 is as shown in (6.7).

$$r_2 = q_2 \cdot 2^2 + x_2 x_0 \cdot 2^2 + x_0 x_2 \cdot 2^2 = q_2 \cdot 2^2 + x_2 x_0 \cdot 2^3 = q_2 \cdot 2^2 \tag{6.7}$$

The computations of the remaining factors in the corresponding vectors are in the same manner, as shown in Table 6.1.

Table 6.1 The remaining factors in the corresponding vectors.

factor	value
r_3	$q_3 \cdot 2^3 + x_2x_0 \cdot 2^3$
r_4	$x_2 \cdot 2^4 + x_2x_1 \cdot 2^4$
s_3	$r_3 \cdot 2^3$
s_4	$r_4 \cdot 2^4 + x_3x_0 \cdot 2^4$
s_5	$r_5 \cdot 2^5 + x_3x_1 \cdot 2^5$
s_6	$x_3 \cdot 2^6 + x_3x_2 \cdot 2^6$

By applying this simplified squarer unit, plenty of adders saved which leads to the optimization of the hardware architecture.

6.2.3 Truncation and Optimization

Due to floating-point representation, the coefficients, $l_{2,i}$, $j_{2,i}$ and $c_{2,i}$, result in truncation, to limit the word length. By simulating different word lengths, each set of the coefficients, $l_{2,i}$, $j_{2,i}$, and $c_{2,i}$,can be optimized to meet the required output precision.

6.3 Post-processing

The input of the post-processing step y is ranging from 0 to 1. The output of post-processing step z should be in the same range as the target function to meet the requirement. Take sin(x) as a target function for instance, the input x is ranging from 0 to $\pi/2$, which is the same range for y from 0 to 1. Hence, the equation in the post-processing step should be $z = y$.

CHAPTER 7

7 Hardware Architecture Using NR

According to Chapter 5, a LUT is essential to select a proper initial guess β_n that is close enough to the true root value σ. This is to speed up the NR convergence, which results in less number of iterations. A general architecture of NR algorithm is as shown in Figure 7.1.

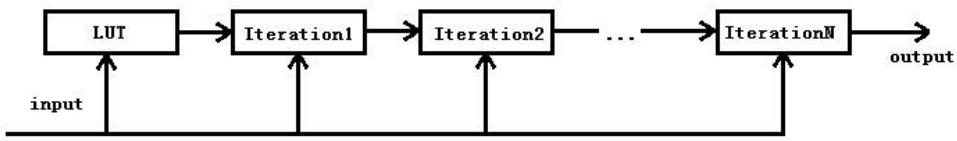

Fig 7.1 The general architecture of NR iterations.

However, in order to meet a specific precision requirement, the deduction of the number of iterations results in a larger LUT size. Hence, there is a trade-off between the size of the LUT and the number of iterations. It is worth mentioning that the function inside every iteration block differs when the target function is altered, which means if the target function changes, the modification for the function inside each iteration block is essential. This is the main drawback of NR iterations.

CHAPTER 8

8 Error Analysis

8.1 Error Behavior Metrics

In order to measure error behavior, there are 5 major factors to be considered, namely, maximum positive and negative absolute error, median error, mean error, standard deviation [14] and root mean square (RMS) error.

8.1.1 Maximum positive and negative error

The maximum positive and negative errors are the most absolute difference and the least absolute difference between the actual value and the approximated value respectively.

8.1.2 Median error

Median error is the value of the sample that lies in the middle position of one set of samples. For an odd set of samples, the median error is the value of the sample that is in the middle position. That is, the number of larger samples equals the number of smaller samples. For an even set of samples, the median error is the average value of the two central samples.

8.1.3 Mean error

Mean error is the average of all the error samples. Each error sample is the difference between the actual value and the approximated value. Assume that the approximated values are expressed as $y_1, y_2, \ldots, y_n$, the actual values are expressed as $x_1, x_2, \ldots, x_n$. Mean error can thus be described according to (8.1).

$$e_{mean} = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i) \tag{8.1}$$

where n is the number of samples.

8.1.4 Standard deviation

Standard deviation is a measure of how are the samples spreading out. That is to say, it is the square root of the variance. The variance defined as the average of the squared differences from the mean value of errors. To sum up, standard deviation is as shown in (8.2).

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}((y_i - x_i) - e_{mean})^2 \tag{8.2}$$

8.1.5 RMS

The RMS error is the square root of the mean square error. Mean square error is the average of the squares of the difference between approximated values and actual values, as described in (8.3).

$$e_{rms} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2} \tag{8.3}$$

8.2 Error Distribution

Error distribution is an alternative factor to consider. It illustrates the possibilities of each value of error that occurs. Figure 8.1 demonstrates an example of error distribution.

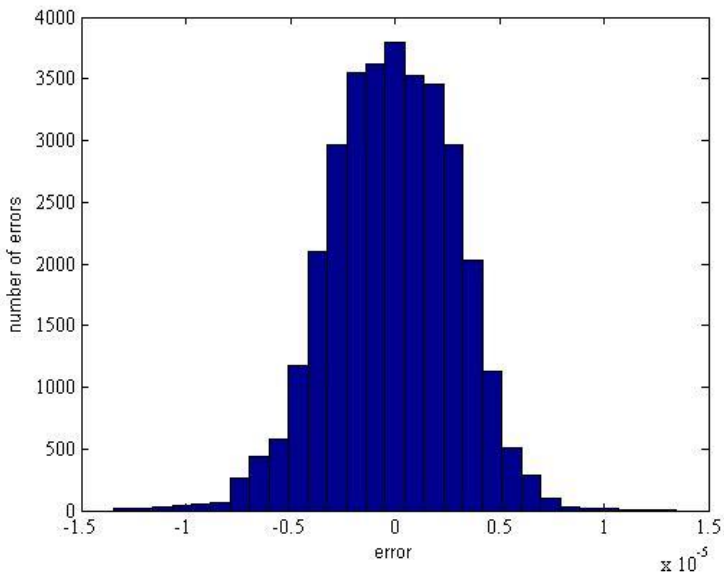

Fig 8.1 An example of error distribution.

As shown in Figure 8.1, the *x*-axis represents the value of the error and the *y*-axis stands for the number of errors that appear. Figure 8.1 is close to a Gaussian distribution, which means the possibilities of errors are high in the center and decrease on the sides.

CHAPTER 9

9 Implementation of Number Inversion applying HPS and NR iteration

In this chapter, the implementation of the number inversion architecture is applicable for two different methods. For the HPS method, the implementations for three different architectures with different intervals are realizable in hardware. For the NR method, the implementation is realizable for one and two iterations respectively. The common input that is to be converted is in the interval (1, 2) with a 15-bit mantissa and an output precision of 16 bits.

9.1 HPS

Based on the HPS method mentioned in Chapter 4, the implementation of the number inversion function is realizable in hardware. The coefficients of the sub-functions, $s_1(x)$ and $s_2(x)$, are selected according to the method mentioned in Section 4.1 and Section 4.2. Furthermore, the hardware implementation consists of three different architectures with different number of intervals, namely 16, 32, and 64 intervals, when developing the second sub-function, $s_2(x)$.

9.1.1 Development of Sub-functions

As mentioned in Section 4.1, the normalization of the original function, $f_{org}(x)$, is essential, to the range $0 \leq x < 1$ on the x-axis and $0 \leq y < 1$ on the y-axis. The number inversion function, shown in (9.1).

$$f(x) = \frac{1}{x} \tag{9.1}$$

9.1.1.1 Pre-processing

As mentioned in Chapter 2, the mantissa, v, is ranging from 1 to 2. In the pre-processing part, the range of the input to the processing part x is subtracted with 1 as shown in (9.2).

$$x = v - 1 \tag{9.2}$$

9.1.1.2 Processing

Initially, the normalization of the original function, $f_{org}(x)$, is essential, to the range $0 \leq x < 1$ on the *x*-axis and $0 \leq y < 1$ on the *y*-axis. The original function, $f_{org}(x)$, shown in (9.3).

$$f_{org}(x) = \frac{2}{1+x} - 1 = \frac{1-x}{1+x} \tag{9.3}$$

Figure 9.1 demonstrates the graphs of the original function, $f_{org}(x)$, and the target function, $f(x)$.

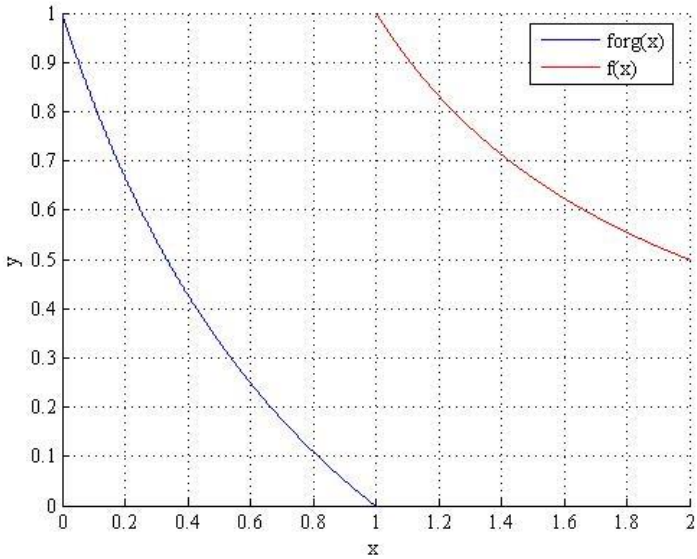

Fig 9.1 The graphs of the original function, $f_{org}(x)$, and the target function, $f(x)$.

9.1.1.3 Post-processing

In the post-processing part, divide the output *y* by 2. Since the output *y* from the processing part is from $0 < z \leq 1$, the output *z* from the post-processing part has to be in the range from $0.5 < z \leq 1$. The post-processing is as illustrated in (9.4).

$$z = \frac{y+1}{2} \tag{9.4}$$

The pre-processing, processing, and post-processing parts result in the target function, $f(x)$.

9.1.1.4 First Sub-function

The first sub-function, $s_1(x)$, is a second order polynomial, defined in (9.5).

$$s_1(x) = l_1 + k_1 x + c_1(x - x^2) \tag{9.5}$$

As shown in Figure 9.1, the first sub-function, $s_1(x)$, should cut the same starting point and ending point as the original function, $f_{org}(x)$, since it is to approach $f_{org}(x)$. The starting point is (0, 1) which gives l_1 to be 1. The coefficient k_1, calculated to be -1, based on both the starting point and the ending point. The first sub-function, $s_1(x)$, is thus simplified according to (9.6).

$$s_1(x) = 1 - x + c_1(x - x^2) \tag{9.6}$$

9.1.1.5 Selection of a proper c_1

Regarding Section 4.3 and 4.5, the combination of c_1 and the number of intervals divided by the second sub-function $s_2(x)$ is important to select a proper c_1. Figure 9.2 demonstrates the precision function of c_1, sweeping from -0.8 to 0, combined with different number of intervals in the second sub-function, $s_2(x)$.

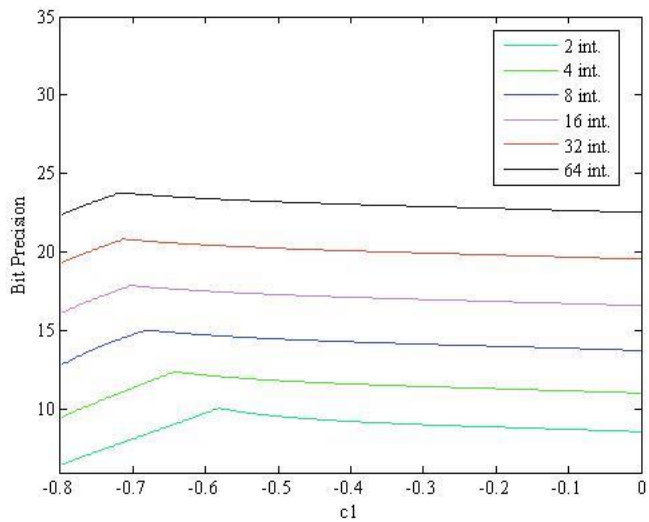

Fig 9.2 The precision function for c_1, combined with 2, 4, 8, 16, 32, 64 intervals in the second sub-function.

In order to gain a hardware-friendly architecture, c_1 is selected to 0. Furthermore, to meet the output precision requirements of 16 bits, the numbers of intervals of the second sub-function, $s_2(x)$, are selected to 16, 32, and 64.

Since c_1 is selected to 0, the first sub-function $s_1(x)$, is simplified according to (9.7).

$$s_1(x) = 1 - x \tag{9.7}$$

The first help function, $f_1(x)$, is the quotient of the original function, $f_{org}(x)$, and the first sub-function, $s_1(x)$, described in (9.8).

$$f_1(x) = \frac{f_{org}(x)}{s_1(x)} = \frac{1}{1 + x} \tag{9.8}$$

9.1.1.6 Second sub-function

According to Section 4.2, based on the number of intervals, the different sets of coefficients, $l_{2,i}$, $j_{2,i}$, and $c_{2,i}$, of the second sub-function, $s_2(x)$, are developed. They are as listed in Table A.1 to A.9 in Appendix A.1.

9.1.2 Hardware architecture

Figure 9.3 demonstrates the hardware architecture of the HPS methodology for the number inversion function.

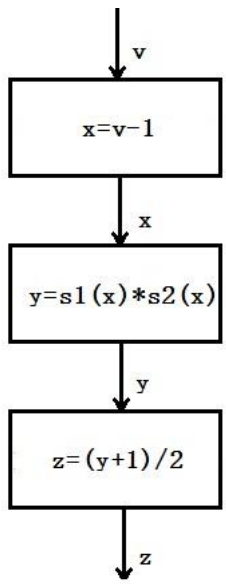

Fig 9.3 The hardware architecture of the number inversion function.

It consists of three steps, pre-processing, processing, and post-processing. The pre-processing step is the normalization part in the algorithm. In the processing step, two sub-functions are required to approximate the target function. In the post-processing step, the values (with normalized input) obtained from the previous step are translated into true values. These three steps are in Section 9.1.1.1, 9.1.1.2, and 9.1.1.3 in detail.

Based on Figure 9.3, a detailed block diagram of the different parts of the implementation of the number inversion function is as shown in Figure 9.4. As described in Chapter 2, the input is a floating-point number with exponent part and mantissa part. The computation for the mantissa and the exponent is separate. For the exponent, the sign bit changes depending on the result. Concerning the mantissa, if the mantissa of the result is less than 1, the exponent has to be subtracted with 1. In addition, if the mantissa is less than 1, multiply the mantissa with 2. In the approximation part, the mentioned three steps are used.

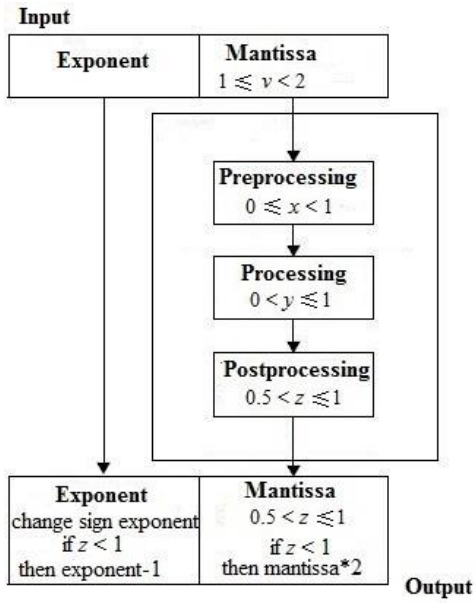

Fig 9.4 The block diagram of the procedure of number inversion function.

9.1.2.1 The optimized hardware architectures

The optimized hardware architectures of three different number of intervals in the second sub-function, $s_2(x)$, are demonstrated in Figure 9.5, Figure 9.6, and Figure 9.7. They are architectures using HPS method with 16, 32 and 64 intervals. In the corresponding s_2 parts in the figures, the size of the LUT ($l_{2,i}$, $j_{2,i}$, and $c_{2,i}$, in the figures) increases when the number of intervals increases in each architecture. The needed word lengths to meet the required precision become smaller, thus, result in smaller multipliers, adders and squarer. Among these three architectures, the 64-interval structure claims the largest size of the LUT and the smallest multipliers, adders and squarer.

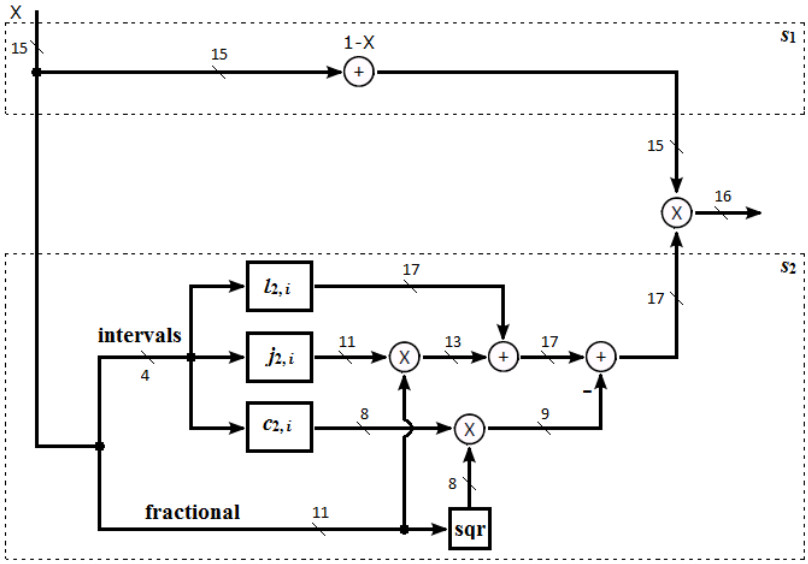

Fig 9.5 The hardware architecture for 16-interval structure.

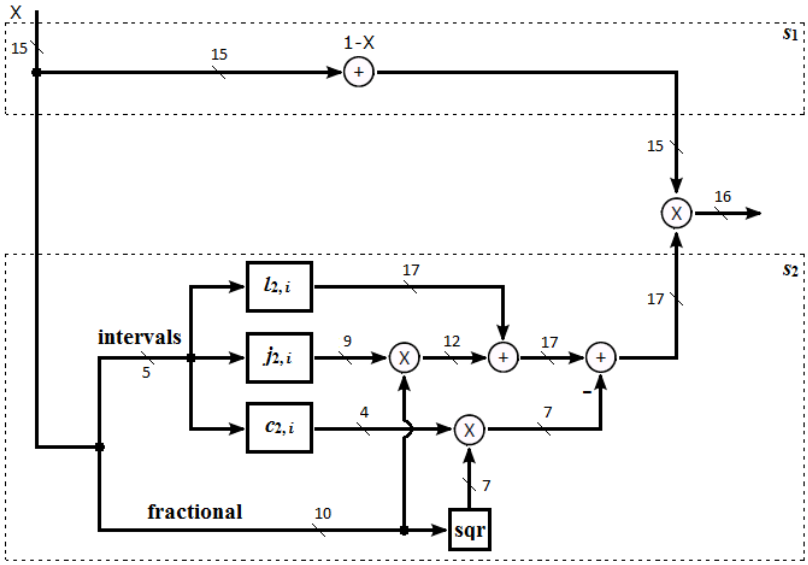

Fig 9.6 The hardware architecture for 32-interval structure.

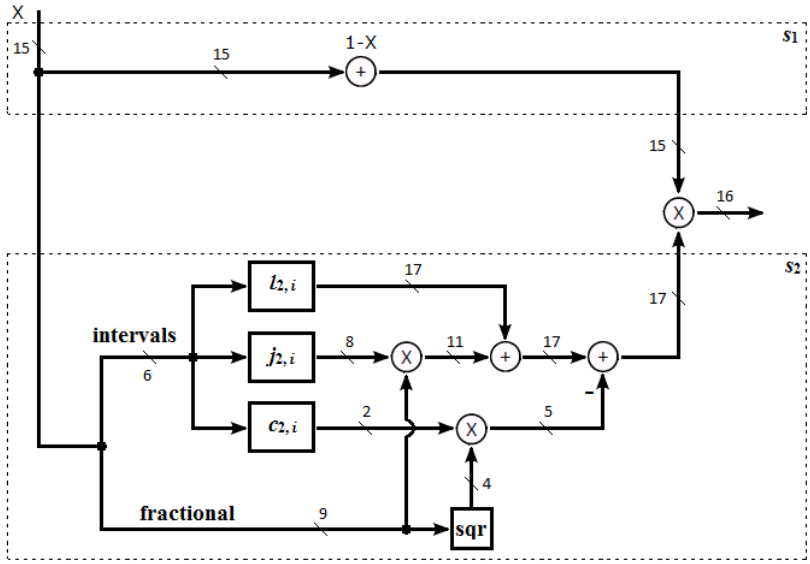

Fig 9.7 The hardware architecture for 64-interval structure.

9.2 NR Iteration

This section contains the introduction of one-stage NR iteration and two-stage NR iteration.

9.2.1 Initial Guess

As mentioned in Chapter 7, in order to achieve a quick convergence, a good initial guess is crucial. If the number of stages is only one, a large LUT containing different initial guesses for the different intervals is required. Furthermore, along with the growth of the number of stages, the reduction of the size of LUT is significant. Based on equation (5.9), two LUTs after truncation and optimization regarding one-stage iteration and two-stage iteration, listed according to Table A.10 and A. 11 in Appendix A.2.

9.2.2 Hardware Architecture

The diagrams in Figure 9.8 and Figure 9.9 demonstrate the architectures using NR method for a single iteration and two iterations (unrolled) with needed word lengths to meet the required output precision. Compare to the architecture for two iterations, the one-stage iteration architecture claims a larger LUT size, but a smaller number of components.

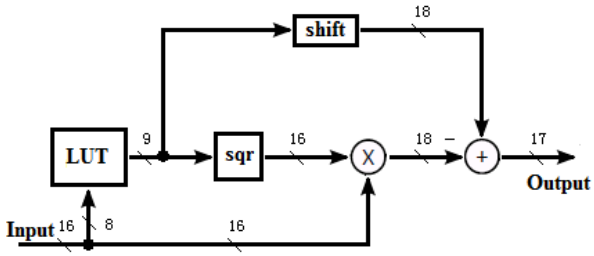

Fig 9.8 The hardware architecture for one-stage NR iteration architecture.

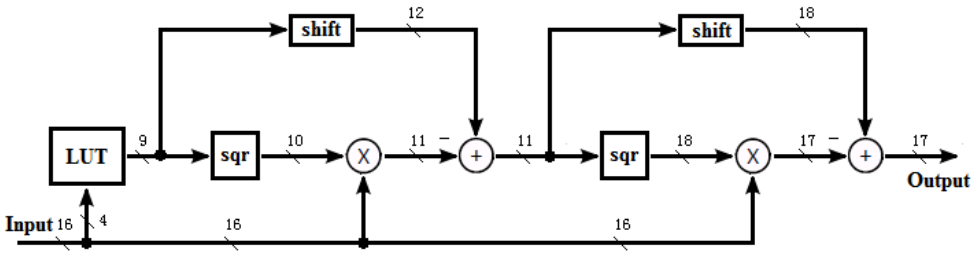

Fig 9.9 The hardware architecture for two-stage NR iteration architecture.

CHAPTER 10

10 Implementation Results

The implementation results consist of error behavior, error metrics, chip area, timing, and power estimation. By applying a 65nm LPHVT COMS technology, the implementation results include these aspects for all hardware architectures. All of the conditions are at normal case and at room temperature.

10.1Error Behavior

Figure 10.1 represents the error behavior for the 16-interval structure in the second sub-function, $s_2(x)$.

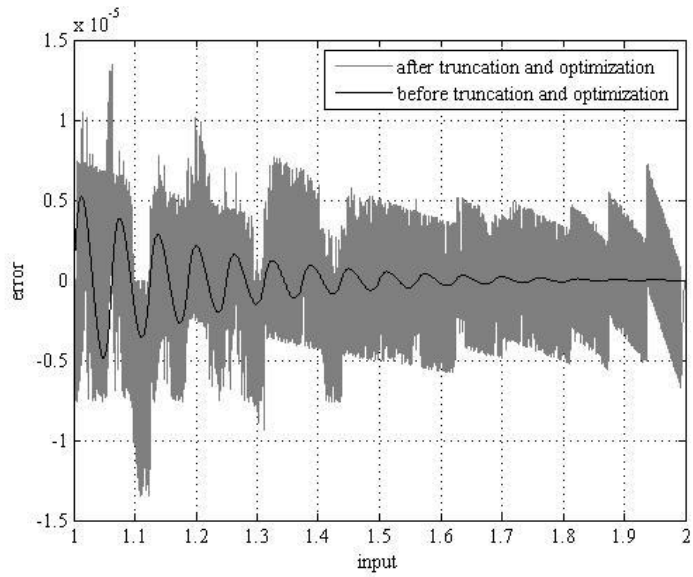

Fig 10.1 The error behavior before and after truncation and optimization applying 16-interval structure.

As shown in Figure 10.1, the black and grey curves represent the error before and after the truncation and optimization respectively. The errors are lying equally around 0.

Figure 10.2 demonstrates the error in Figure 10.1 in bit unit.

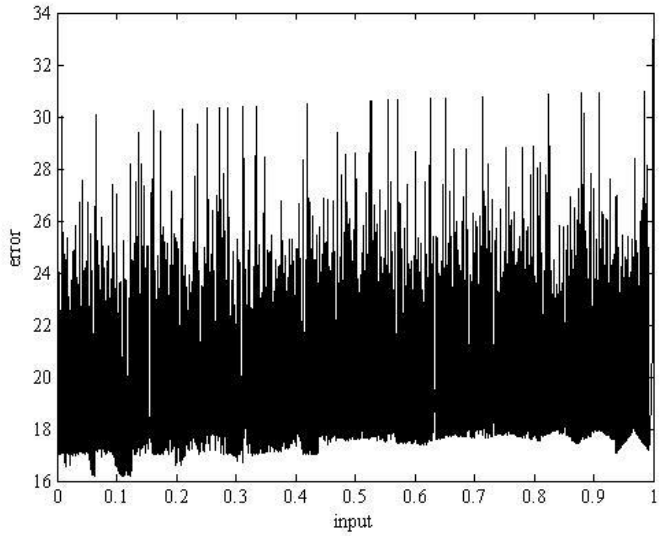

Fig 10.2 The error of Fig 10.1 in bit unit.

According to Figure 10.2, the 16-bit target precision requirement is satisfied. Figure 10.3 describes the error distribution in Figure 10.1.

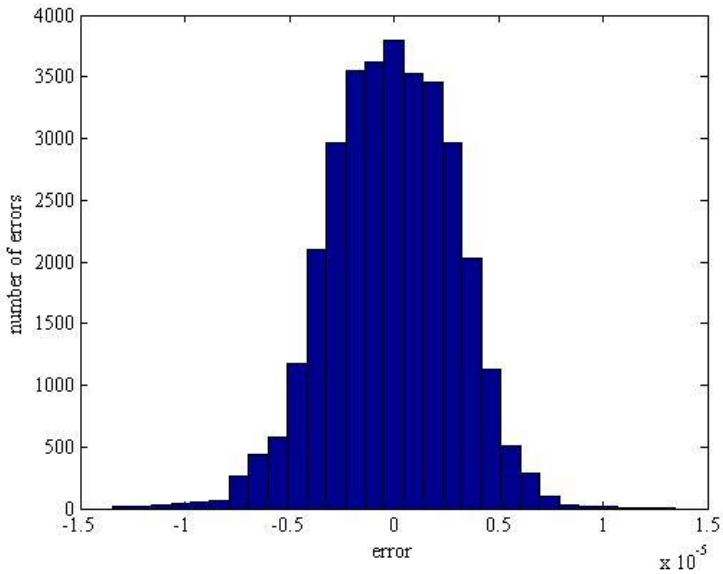

Fig 10.3 The error distribution based on Fig 10.1.

As shown in Figure 10.3, the error is in the interval of ($-1.5\cdot10^{-5}$, $1.5\cdot10^{-5}$). In general, it realizes a Gaussian distribution, which represents a probability distribution.

The error distribution of 32, and 64 intervals in the second sub-function, $s_2(x)$, can be realized in the same manner, as shown in Figure 10.4 and Figure 10.5.

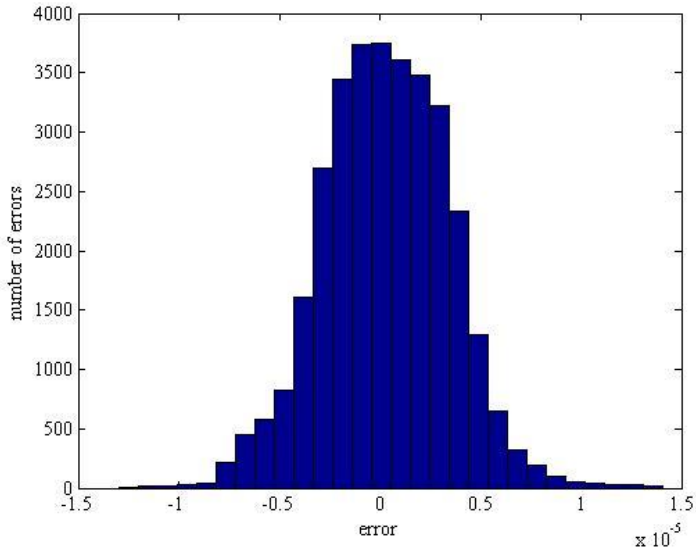

Fig 10.4 The error distribution for 32-interval structure.

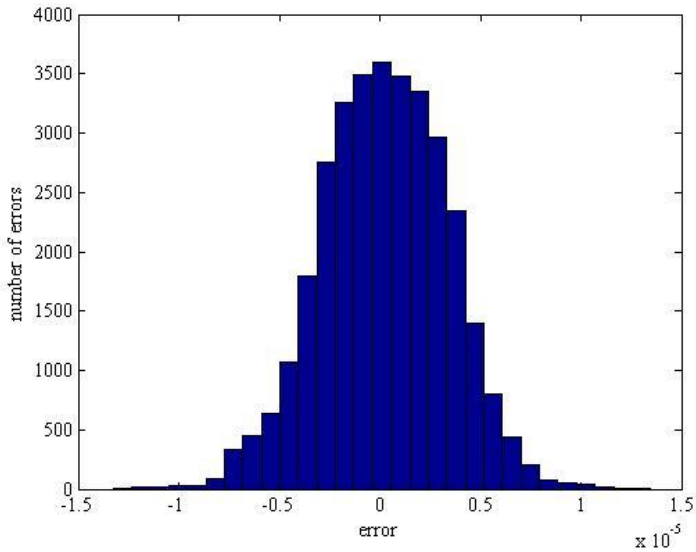

Fig 10.5 The error distribution for 64-interval structure.

According to Figure 10.3, Figure 10.4, and Figure 10.5, along with the growth of the number of intervals in the second sub-function, $s_2(x)$, the numbers of errors slightly decrease around zero.

Figure 10.6 and demonstrates error behavior after truncation and optimization applying one-stage NR iteration. The errors are lying equally around 0. Figure 10.7 illustrates error behavior before truncation and optimization. The errors are not lying around 0. It is not readable when placing these two figures together.

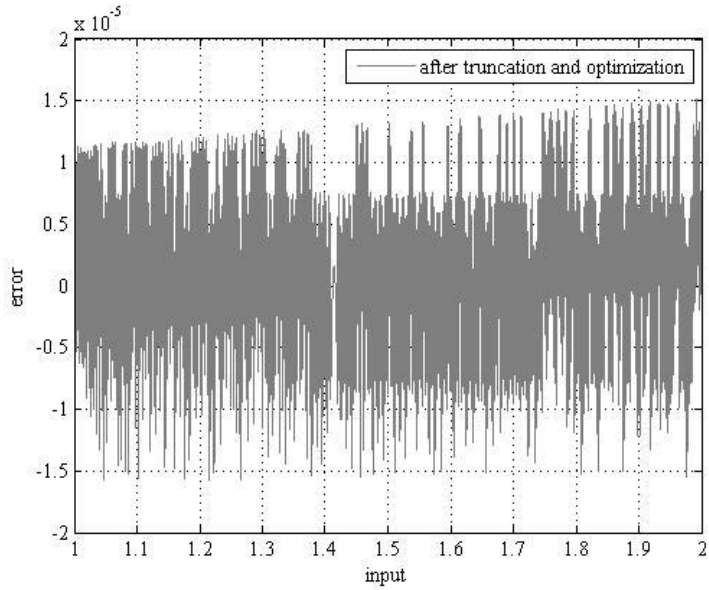

Fig 10.6 The error behavior after truncation and optimization applying one-stage NR iteration.

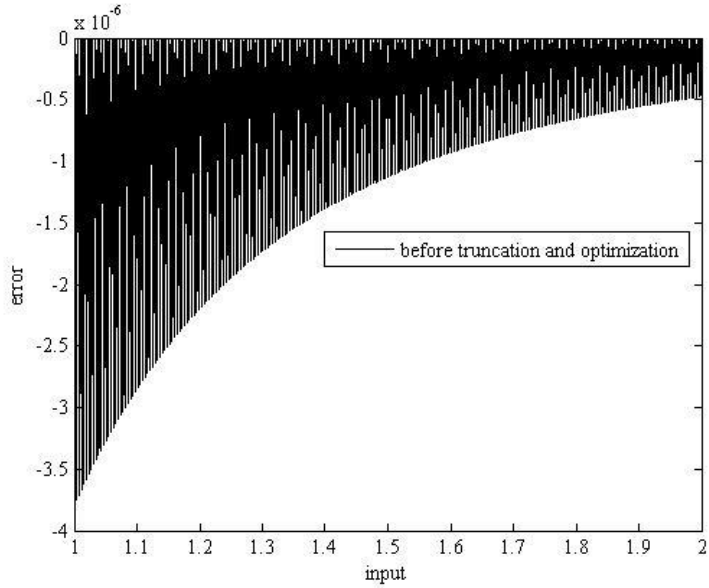

Fig 10.7 The error behavior before truncation and optimization applying one-stage NR iteration.

Figure 10.8 demonstrates the error in Figure 10.6 in bit unit.

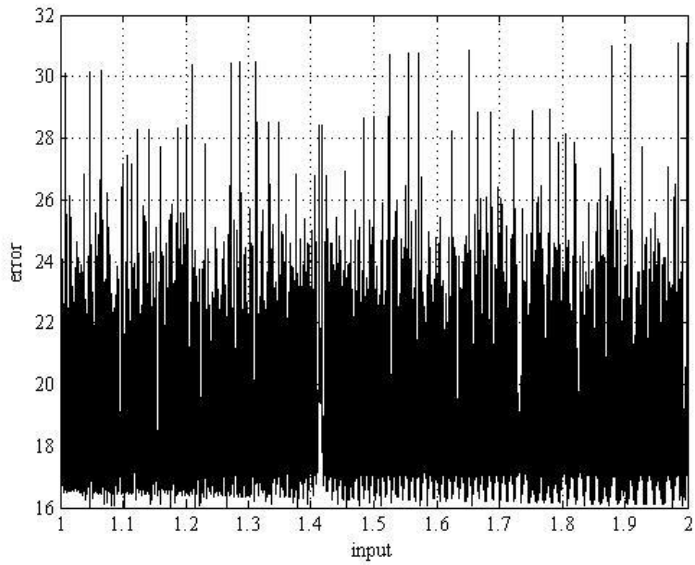

Fig 10.8 The error in Fig 10.6 in bit unit.

According to Figure 10.8, the 16-bit target precision requirement is realizable. The error distribution is as described in Figure 10.9 based on Figure 10.6.

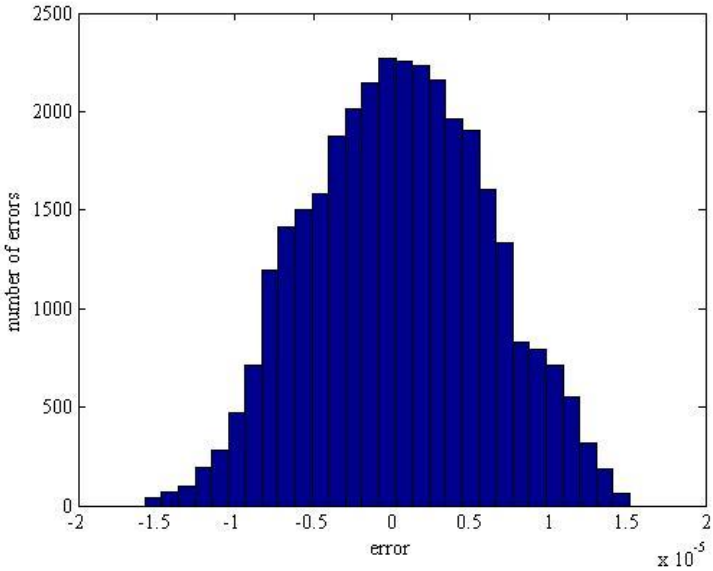

Fig 10.9 The error distribution based on Fig 10.6.

As shown in Figure 10.9, the error is in the interval ($-1.5\cdot10^{-5}$, $1.5\cdot10^{-5}$). In general, it realizes a Gaussian distribution, which is a probability distribution.

The error distribution of the two-stage NR iteration is realizable in the same manner, as shown in Figure 10.10.

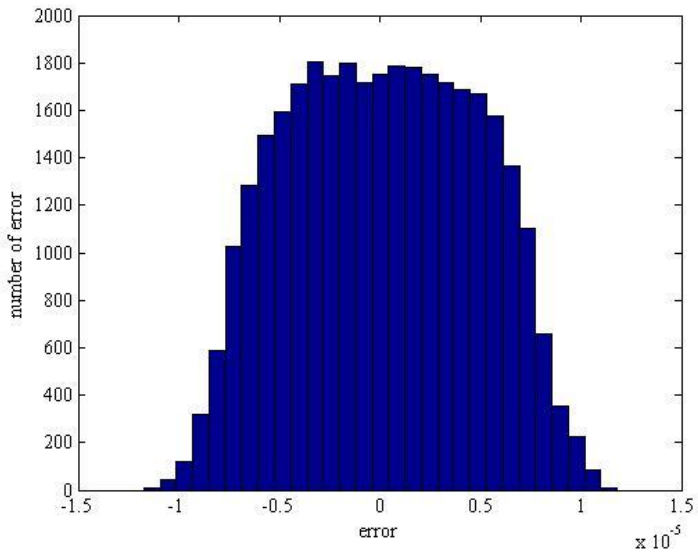

Fig 10.10 The error distribution of two-stage NR iteration

According to Figure 10.9 and Figure 10.10, along with the growth of the number of iterations, the reduction of the size of the LUT is significant. This results in that the error distribution becomes wider. As mentioned in Chapter 7, there is a trade-off between the size of LUT and the number of NR iterations.

To close this section, the HPS architectures claim a better error distribution than the NR iteration architectures. The error distributions for both NR iteration architectures are much wider. The most of the errors for 16-interval structure applying HPS are lying in the interval ($-0.5\cdot10^{-5}$, $0.5\cdot10^{-5}$) according to Figure 10.1. However, with respect to Figure 10.6, the most of the errors for one-stage NR iteration architecture are lying in the interval ($-1\cdot10^{-5}$, $1\cdot10^{-5}$). This indicates that the 16-interval structure claims a better error behavior than one-stage NR iteration architecture.

10.2 Error metrics

Table 10.1 illustrates the error metrics for five architectures after truncation and optimization.

Table 10.1 Error metrics for five architectures after truncation and optimization.

Error type	16HPS (10^{-6}/bits)	32HPS (10^{-6}/bits)	64HPS (10^{-6}/bits)	1NR (10^{-6}/bits)	2NR (10^{-6}/bits)
Maximum positive error	13.5/16.24	14.0/16.17	13.5/16.24	15.7/16.06	11.8/16.43
Maximum negative error	13.4/16.24	13.0/16.29	13.3/16.26	15.2/16.01	11.7/16.43
Mean error	2.40/18.73	2.52/18.66	2.53/18.66	4.65/17.77	3.96/18.01
Median error	2.11/18.92	2.18/18.87	2.19/18.87	4.12/17.95	3.77/18.08
Standard deviation	2.99	3.15	3.15	5.67	4.66
RMS	3.00	3.16	3.16	5.69	4.66

According to Table 10.1, there are slight differences for each error factor. Due to Gaussian distribution, the differences between the Standard deviation and the

RMS in each metrics are small as well. In general, the HPS architectures claim the smaller errors concerning all the error types above.

10.3 Synthesis Constraints

There are two synthesis constraints, high-speed constraint and low-area constraint. For a high-speed circuit, a specific high clock frequency and 0-area constraint are set. In order to realize a high-speed constraint, a small clock period is set in the synthesis script to maintain a high clock frequency. In order to realize an area optimized circuit, the max chip area in the synthesis script is set to zero and a low clock frequency (high clock period) is in specific.

10.4 Chip Area

Table 10.2 demonstrates the minimum chip area for five hardware architectures. To be clear, 16HPS stands for 16 intervals in the second sub-function, $s_2(x)$, for HPS method, 1NR stands for one stage iteration for the NR method. The representations of the remaining architectures are in the same manner.

Table 10.2 Chip area when minimum chip area constraint is set.

Architecture	Chip Area (combinational, μm^2)	(%)	Chip Area (total, μm^2)	(%)
16HPS	5599	98	5857	98
32HPS	4956	87	5213	87
64HPS	4759	83	5016	84
1NR	4106	72	4364	73
2NR	5710	100	5968	100

To synthesis the design, we add registers. This results in the difference between the combinational chip area and total chip area.

By setting no chip area constraints, the maximum chip area for five architectures are as listed according to Table 10.3.

Table 10.3 Chip area when maximum speed constraint is set.

Architecture	Chip Area (combinational, μm^2)	(%)	Chip Area (total, μm^2)	(%)
16HPS	9328	81	9590	82
32HPS	7996	70	8257	70
64HPS	7828	68	8086	69
1NR	7079	62	7337	63
2NR	11455	100	11729	100

Based on Table 10.2 and Table 10.3, the two-stage NR iterations architecture claims the largest chip area. On the contrary, the one-stage NR iteration architecture claims the least chip area. Along with the growth of the number of intervals in the second sub-function, $s_2(x)$, applying HPS, the chip area shows a tendency to decline.

10.5 Timing

Table 10.4 illustrates timing report for high-speed circuits. The general idea is to locate the minimum clock period when maintaining a minimum slack. Therefore, the minimum clock period represents the highest possible clock frequency.

Table 10.4 Timing report when maximum speed constraint is set.

Architecture	Critical Path/Frequency (combinational, ns/MHz)	Frequency (%)	Critical Path/Frequency (total, ns/MHz)	Frequency (%)
16HPS	4.53/221	138	4.85/206	136
32HPS	3.92/255	159	4.27/234	154
64HPS	3.46/289	180	3.77/265	174
1NR	3.55/282	176	3.88/258	170
2NR	6.25/160	100	6.56/152	100

Table 10.5 demonstrates timing report for low-area circuits by setting the max chip area in the synthesis script to 0.

Table 10.5 Timing report when minimum chip area constraint is set.

Architecture	Critical Path/Frequency (combinational, ns/MHz)	Frequency (%)	Critical Path/Frequency (total, ns/MHz)	Frequency (%)
16HPS	11.15/87	138	11.61/86	139
32HPS	10.00/100	159	10.39/96	155
64HPS	8.91/112	178	9.30/108	174
1NR	9.60/104	165	10.02/100	161
2NR	15.77/63	100	16.21/62	100

According to Table 10.4 and Table 10.5, the two-stage NR iterations architecture claims the largest critical path. However, 64 intervals HPS architecture claims the least critical path. Along with the growth of the number of intervals in the second sub-function, $s_2(x)$, applying HPS, the critical path shows a tendency of decline as well.

10.6 Power estimation

The power in CMOS transistors consists of dynamic power and static power, as shown in (10.1).

$$P_{total} = P_{dynamic} + P_{static} \tag{10.1}$$

The static power is the power consumed when there is no circuit activity. It is due to sub threshold currents, reverse bias leakage gate leakage, and another few currents [15]. The static power consumption is the product of the device leakage current and the supply voltage. On the contrary, the dynamic power is the power consumed when the inputs are active. It is due to dynamic switching events. It consists of switching power and internal power, as shown in (10.2).

$$P_{dynamic} = P_{switching} + P_{\text{int}ernal} \tag{10.2}$$

According to the tool vendor, "The switching power is determined by the capacitive load and the frequency of the logic transitions on a cell output" – quoted from [16]. Moreover, "The internal power is caused by the charging of internal loads as well as by the short-circuit current between the N and P transistors of a gate when both are on" – quoted from [16]. The dynamic power is thus as expressed according to (10.3).

$$P_{dynamic} = aCV^2 f \tag{10.3}$$

where a is the activity factor, C is the switched capacitance, V is the supply voltage, and f is the clock frequency.

Figure 10.11 demonstrates the power consumption for 16, 32, and 64 intervals in the second sub-function, $s_2(x)$, applying HPS as well as one-stage, and two-stage NR iteration architectures at multiple clock frequencies. All the power consumption values are as listed in Table A.12 to Table A.16 in Appendix A.3.

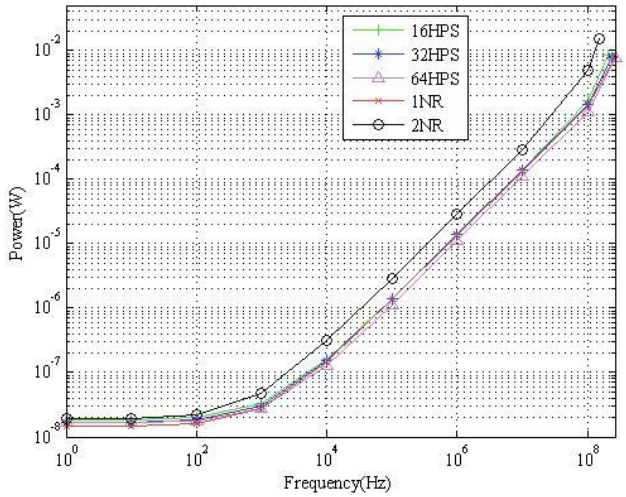

Fig 10.11 The power consumption for five architectures at multiple clock frequencies.

Figure 10.12 demonstrates the zoomed-in figure of Figure 10.1.

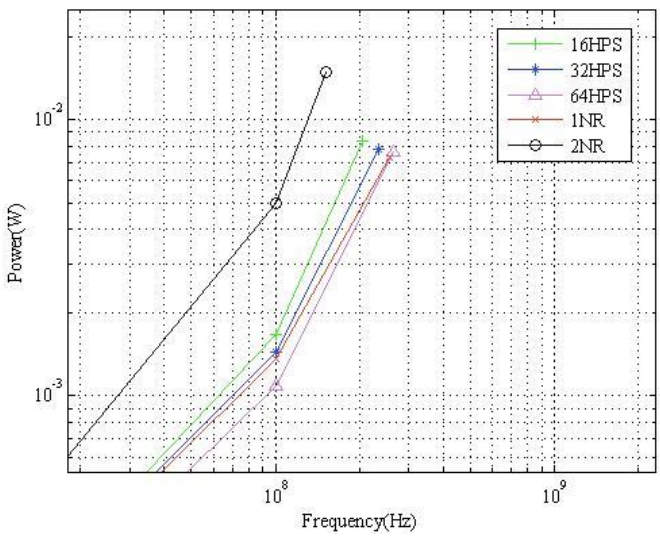

Fig 10.12 The zoomed-in figure of Fig 10.1.

In Figure 10.11 and Figure 10.12, the two-stage NR iteration architecture claims the highest power consumption. On the contrary, the 64-interval structure applying HPS method claims the lowest power consumption. All five curves are close in Figure 10.11. The last point is at the highest possible frequency. However, when zooming in Figure 10.11 partially, there are slight differences. When applying

HPS, the power consumption is less along with the growth of the number of intervals in the second sub-function, $s_2(x)$. When applying the NR iterations, the power consumption is increasing along with the growth of the number of iterations due to a large reduction of the LUT size. In general, one-stage NR iteration consumes less power, two-stage NR iterations consumes more power.

Figure 10.13 illustrates the switching power, internal power, static power and total power for 64 intervals in the second sub-function, $s_2(x)$, applying HPS after post synthesis simulation at multiple clock frequencies.

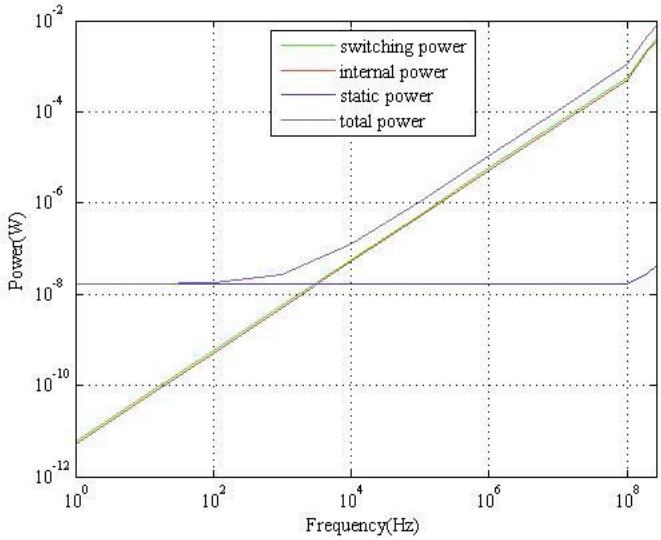

Fig 10.13 The switching power, internal power, static power and total power for 64 interval structure applying HPS after post synthesis simulation at multiple clock frequencies.

Figure 10.14 illustrates the switching power, internal power, static power and total power for one-stage NR iteration architecture after post synthesis simulation at multiple clock frequencies.

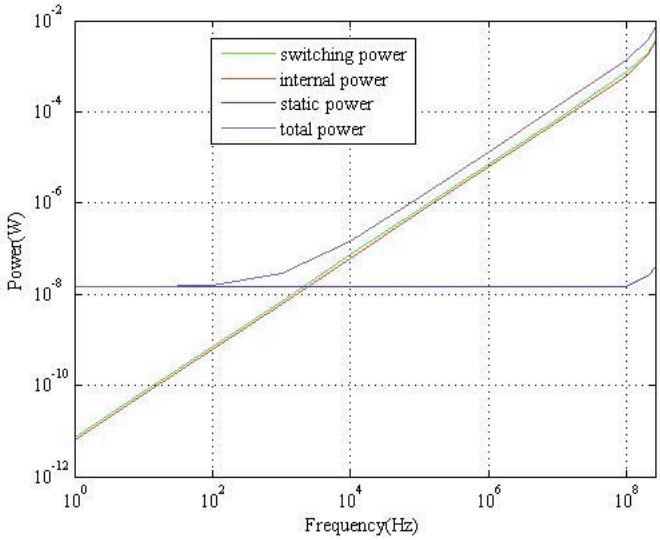

Fig 10.14 The switching power, internal power, static power and total power applying one-stage NR iteration after post synthesis simulation at multiple clock frequencies.

The green curve is switching power, the red curve is internal power, the blue curve is static power, and the grey curve is total power. The dynamic power is the difference between total power and static power, which is absent in the figure due to the closeness between total power and dynamic power. Comparing Figure 10.13 to Figure 10.14, all curves of power are close. However, the switching power and the internal power in 64 intervals applying HPS are slightly less than one-stage NR iteration.

Figure 10.15 and Figure 10.16 demonstrates the switching power, internal power, static power and total power for 64 intervals in the second sub-function, $s_2(x)$, applying HPS and one-stage NR iteration after post layout simulation at multiple clock frequencies respectively.

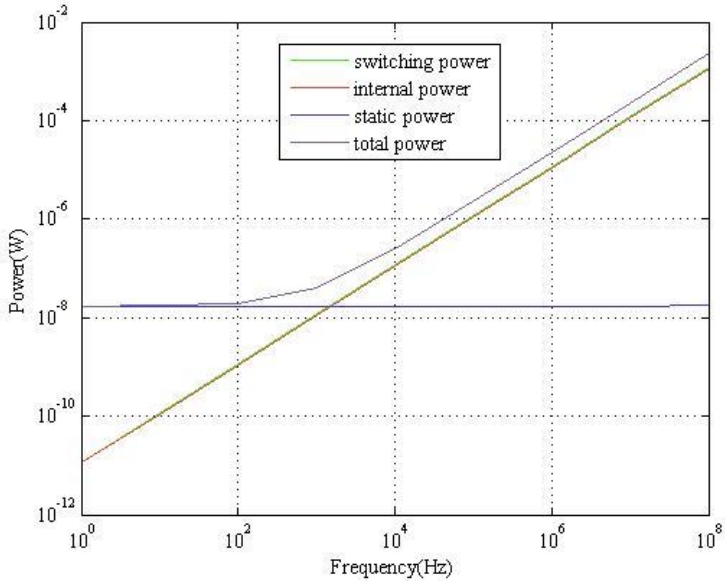

Fig 10.15 The switching power, internal power, static power and total power for 64 interval structure applying HPS after post layout simulation at multiple clock frequencies.

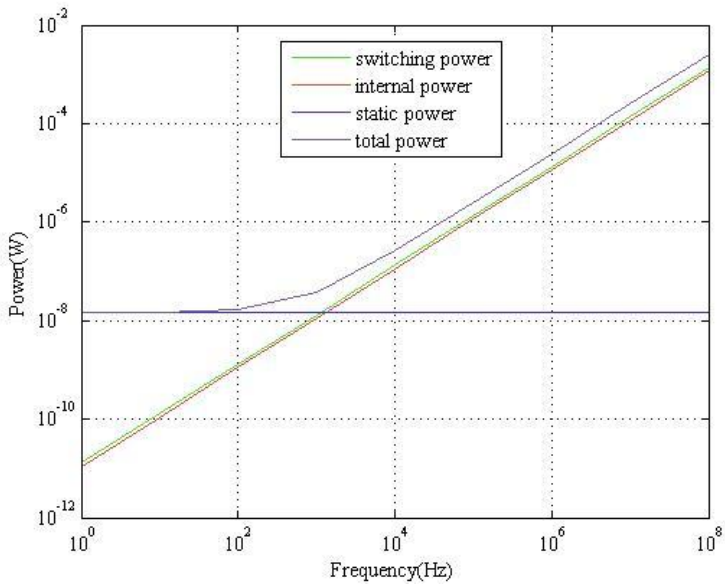

Fig 10.16 The switching power, internal power, static power and total power applying one-stage NR iteration after post layout simulation at multiple clock frequencies.

The power consumption after post layout simulation is larger than that after post synthesis simulation at corresponding frequencies for both architectures.

10.7 Physical Layout

Figure 10.17 and Figure 10.18 demonstrates the physical layout of two different architectures.

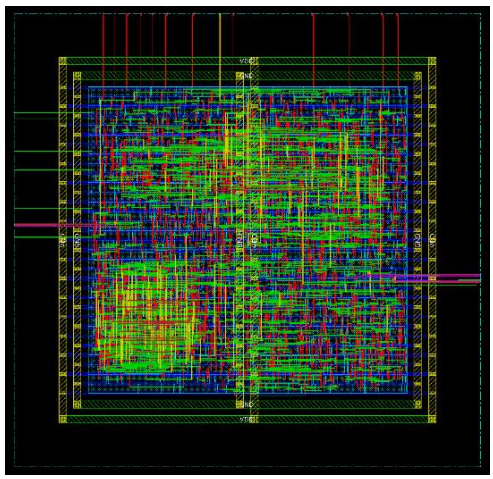

Fig 10.17 The physical layout of 64 interval structure applying HPS method.

After P&R, the core size is 86.3 (width) multiplied by 83.2 (height), which is 7180 um^2 and the die size is 126.3 (width) multiplied by 123.2 (height), which is 15560 um^2. The die size is, however, not relevant since there are no pads. According to Table 10.1, the minimum chip area of 64HPS architecture is 4759 um^2. The core utilization is 99.86%.

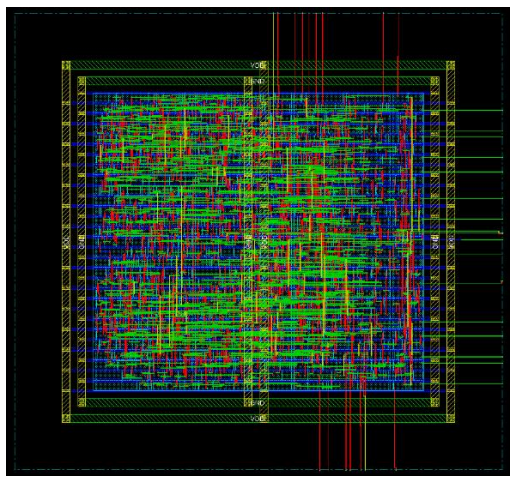

Fig 10.18 The physical layout of one-stage NR iteration architecture.

The core size is 83.3(width) multiplied by 75.4 (height), which is 6281 um^2 and the die size is 123.3 (width) multiplied by 115.4 (height), which is 14229 um^2. The theoretical minimum chip area of this architecture in Table 10.1 is smaller than that of 64HPS architecture. It is also true for the actual core size. The core utilization is 99.90%.

CHAPTER 11

11 Conclusions

11.1 Comparison

In this thesis, the implementations for three different architectures applying HPS and two different architectures applying NR iteration are realizable. For the HPS method, the growth of the number of intervals in the second sub-function, $s_2(x)$, results in less chip area, smaller critical path, and less power consumption. In order to realize a simple hardware with maintained accuracy, the fixed coefficients are essential. For NR iteration, a lager LUT gives fewer iterations, but larger chip area, larger critical path, and significant larger power consumption. This is due to that a significantly larger LUT size achieves significantly quicker convergence. The major merits of HPS method compared to NR iterations, listed as follows:

Multiple target function: Plenty of unary functions such as sine, logarithmic, exponential, reciprocal and even square reciprocal is realizable applying HPS method. On the contrary, the amount of the implementations of different functions by NR iterations is far more less.

High flexibility: For the HPS method, different target functions have the similar hardware architecture. The differences are the sets of the coefficients. This is not applicable for NR iterations. For NR iterations, when changing the target function, the modification of the hardware architecture is essential.

11.2 Future Work

For the HPS method, it is inevitable that along with the growth of the number of intervals in the second sub-function, $s_2(x)$, the output precision will be better. What is more, the coefficients in the second sub-function can be deeper optimized. Finally, the modification for the architectures is applicable to meet a better performance.

For the NR iteration method, it is possible to scale down the size of LUT by half in one-stage NR iteration.

Appendix A

A.1 The coefficients in the second sub-function using HPS method.

Table A.1 The coefficients $l_{2,i}$ of 16-interval structure.

coefficient	value	coefficient	value
$l_{2,0}$	1.000000000000000	$l_{2,8}$	0.666679382324219
$l_{2,1}$	0.941184997558594	$l_{2,9}$	0.640014648437500
$l_{2,2}$	0.888893127441406	$l_{2,10}$	0.615409851074219
$l_{2,3}$	0.842117309570313	$l_{2,11}$	0.592613220214844
$l_{2,4}$	0.800010681152344	$l_{2,12}$	0.571456909179688
$l_{2,5}$	0.761917114257813	$l_{2,13}$	0.551773071289063
$l_{2,6}$	0.727287292480469	$l_{2,14}$	0.533416748046875
$l_{2,7}$	0.695663452148438	$l_{2,15}$	0.516357421875000

Table A.2 The coefficients $j_{2,i}$ of 16-interval structure.

coefficient	value	coefficient	value
$j_{2,0}$	0.062347412109375	$j_{2,8}$	0.027740478515625
$j_{2,1}$	0.055267333984375	$j_{2,9}$	0.025573730468750
$j_{2,2}$	0.049285888671875	$j_{2,10}$	0.023651123046875
$j_{2,3}$	0.044250488281250	$j_{2,11}$	0.021911621093750
$j_{2,4}$	0.039947509765625	$j_{2,12}$	0.020385742187500
$j_{2,5}$	0.036224365234375	$j_{2,13}$	0.019012451171875
$j_{2,6}$	0.033020019531250	$j_{2,14}$	0.017761230468750
$j_{2,7}$	0.030212402343750	$j_{2,15}$	0.016632080078125

Table A.3 The coefficients $c_{2,i}$ of 16-interval structure.

coefficient	value	coefficient	value
$c_{2,0}$	0.003555297851563	$c_{2,8}$	0.001083374023438
$c_{2,1}$	0.002975463867188	$c_{2,9}$	0.000961303710938
$c_{2,2}$	0.002517700195313	$c_{2,10}$	0.000854492187500
$c_{2,3}$	0.002151489257813	$c_{2,11}$	0.000762939453125
$c_{2,4}$	0.001846313476563	$c_{2,12}$	0.000686645507813
$c_{2,5}$	0.001602172851563	$c_{2,13}$	0.000610351562500
$c_{2,6}$	0.001388549804688	$c_{2,14}$	0.000549316406250
$c_{2,7}$	0.001235961914063	$c_{2,15}$	0.000503540039063

Table A.4 The coefficients $l_{2,i}$ of 32-interval structure.

coefficient	value	coefficient	value
$l_{2,0}$	1.000000000000000	$l_{2,16}$	0.666679382324219
$l_{2,1}$	0.969696044921875	$l_{2,17}$	0.653076171875000
$l_{2,2}$	0.941192626953125	$l_{2,18}$	0.639999389648438
$l_{2,3}$	0.914283752441406	$l_{2,19}$	0.627471923828125
$l_{2,4}$	0.888885498046875	$l_{2,20}$	0.615394592285156
$l_{2,5}$	0.864868164062500	$l_{2,21}$	0.603790283203125
$l_{2,6}$	0.842109680175781	$l_{2,22}$	0.592605590820313
$l_{2,7}$	0.820518493652344	$l_{2,23}$	0.581840515136719
$l_{2,8}$	0.800018310546875	$l_{2,24}$	0.571479797363281
$l_{2,9}$	0.780479431152344	$l_{2,25}$	0.561447143554688
$l_{2,10}$	0.761901855468750	$l_{2,26}$	0.551757812500000
$l_{2,11}$	0.744194030761719	$l_{2,27}$	0.542419433593750
$l_{2,12}$	0.727279663085938	$l_{2,28}$	0.533393859863281
$l_{2,13}$	0.711112976074219	$l_{2,29}$	0.524642944335938
$l_{2,14}$	0.695648193359375	$l_{2,30}$	0.516288757324219
$l_{2,15}$	0.516357421875000	$l_{2,31}$	0.508323669433594

Table A.5 The coefficients $j_{2,i}$ of 32-interval structure.

coefficient	value	coefficient	value
$j_{2,0}$	0.031188964843750	$j_{2,16}$	0.013854980468750
$j_{2,1}$	0.029357910156250	$j_{2,17}$	0.013305664062500
$j_{2,2}$	0.027648925781250	$j_{2,18}$	0.012756347656250
$j_{2,3}$	0.026062011718750	$j_{2,19}$	0.012268066406250
$j_{2,4}$	0.024658203125000	$j_{2,20}$	0.011779785156250
$j_{2,5}$	0.023315429687500	$j_{2,21}$	0.011352539062500
$j_{2,6}$	0.022094726562500	$j_{2,22}$	0.010925292968750
$j_{2,7}$	0.020996093750000	$j_{2,23}$	0.010559082031250
$j_{2,8}$	0.019958496093750	$j_{2,24}$	0.010192871093750
$j_{2,9}$	0.018981933593750	$j_{2,25}$	0.009826660156250
$j_{2,10}$	0.018066406250000	$j_{2,26}$	0.009460449218750
$j_{2,11}$	0.017272949218750	$j_{2,27}$	0.009155273437500
$j_{2,12}$	0.016479492187500	$j_{2,28}$	0.008850097656250
$j_{2,13}$	0.015747070312500	$j_{2,29}$	0.008483886718750
$j_{2,14}$	0.015075683593750	$j_{2,30}$	0.008300781250000
$j_{2,15}$	0.016632080078125	$j_{2,31}$	0.008056640625000

Table A.6 The coefficients $c_{2,i}$ of 32-interval structure.

Coefficient	Value	Coefficient	Value
$c_{2,0}$	0.000915527343750	$c_{2,16}$	0.000244140625000
$c_{2,1}$	0.000854492187500	$c_{2,17}$	0.000244140625000
$c_{2,2}$	0.000732421875000	$c_{2,18}$	0.000244140625000
$c_{2,3}$	0.000671386718750	$c_{2,19}$	0.000183105468750
$c_{2,4}$	0.000671386718750	$c_{2,20}$	0.000183105468750
$c_{2,5}$	0.000549316406250	$c_{2,21}$	0.000183105468750
$c_{2,6}$	0.000488281250000	$c_{2,22}$	0.000183105468750
$c_{2,7}$	0.000488281250000	$c_{2,23}$	0.000183105468750
$c_{2,8}$	0.000427246093750	$c_{2,24}$	0.000122070312500
$c_{2,9}$	0.000427246093750	$c_{2,25}$	0.000122070312500
$c_{2,10}$	0.000366210937500	$c_{2,26}$	0.000122070312500
$c_{2,11}$	0.000366210937500	$c_{2,27}$	0.000122070312500
$c_{2,12}$	0.000305175781250	$c_{2,28}$	0.000122070312500
$c_{2,13}$	0.000305175781250	$c_{2,29}$	0.000122070312500
$c_{2,14}$	0.000305175781250	$c_{2,30}$	0.000122070312500
$c_{2,15}$	0.000305175781250	$c_{2,31}$	0.000122070312500

Table A.7 The coefficients $l_{2,i}$ of 64-interval structure.

Coefficient	Value	Coefficient	Value
$l_{2,0}$	1.000000000000000	$l_{2,32}$	0.666664123535156
$l_{2,1}$	0.984611511230469	$l_{2,33}$	0.659797668457031
$l_{2,2}$	0.969703674316406	$l_{2,34}$	0.653076171875000
$l_{2,3}$	0.955223083496094	$l_{2,35}$	0.646453857421875
$l_{2,4}$	0.941169738769531	$l_{2,36}$	0.639999389648438
$l_{2,5}$	0.927543640136719	$l_{2,37}$	0.633659362792969
$l_{2,6}$	0.914283752441406	$l_{2,38}$	0.627464294433594
$l_{2,7}$	0.901405334472656	$l_{2,39}$	0.621368408203125
$l_{2,8}$	0.888908386230469	$l_{2,40}$	0.615394592285156
$l_{2,9}$	0.876708984375000	$l_{2,41}$	0.609558105468750
$l_{2,10}$	0.864868164062500	$l_{2,42}$	0.603797912597656
$l_{2,11}$	0.853340148925781	$l_{2,43}$	0.598159790039063
$l_{2,12}$	0.842102050781250	$l_{2,44}$	0.592597961425781
$l_{2,13}$	0.831153869628906	$l_{2,45}$	0.587188720703125
$l_{2,14}$	0.820510864257813	$l_{2,46}$	0.581832885742188
$l_{2,15}$	0.810142517089844	$l_{2,47}$	0.576614379882813
$l_{2,16}$	0.800003051757813	$l_{2,48}$	0.571456909179688
$l_{2,17}$	0.790130615234375	$l_{2,49}$	0.566413879394531
$l_{2,18}$	0.780487060546875	$l_{2,50}$	0.561431884765625
$l_{2,19}$	0.771080017089844	$l_{2,51}$	0.556564331054688
$l_{2,20}$	0.761909484863281	$l_{2,52}$	0.551750183105469
$l_{2,21}$	0.752967834472656	$l_{2,53}$	0.547042846679688
$l_{2,22}$	0.744194030761719	$l_{2,54}$	0.542427062988281
$l_{2,23}$	0.735641479492188	$l_{2,55}$	0.537887573242188
$l_{2,24}$	0.727287292480469	$l_{2,56}$	0.533386230468750
$l_{2,25}$	0.719116210937500	$l_{2,57}$	0.528991699218750
$l_{2,26}$	0.711120605468750	$l_{2,58}$	0.524673461914063
$l_{2,27}$	0.703300476074219	$l_{2,59}$	0.520439147949219
$l_{2,28}$	0.695648193359375	$l_{2,60}$	0.516273498535156
$l_{2,29}$	0.688186645507813	$l_{2,61}$	0.512199401855469
$l_{2,30}$	0.680847167968750	$l_{2,62}$	0.508331298828125
$l_{2,31}$	0.673698425292969	$l_{2,63}$	0.504714965820313

Table A.8 The coefficients $j_{2,i}$ of 64-interval structure.

Coefficient	Value	Coefficient	Value
$j_{2,0}$	0.015563964843750	$j_{2,32}$	0.006896972656250
$j_{2,1}$	0.015075683593750	$j_{2,33}$	0.006774902343750
$j_{2,2}$	0.014648437500000	$j_{2,34}$	0.006652832031250
$j_{2,3}$	0.014221191406250	$j_{2,35}$	0.006469726562500
$j_{2,4}$	0.013793945312500	$j_{2,36}$	0.006347656250000
$j_{2,5}$	0.013427734375000	$j_{2,37}$	0.006225585937500
$j_{2,6}$	0.013000488281250	$j_{2,38}$	0.006103515625000
$j_{2,7}$	0.012634277343750	$j_{2,39}$	0.005981445312500
$j_{2,8}$	0.012329101562500	$j_{2,40}$	0.005859375000000
$j_{2,9}$	0.011962890625000	$j_{2,41}$	0.005798339843750
$j_{2,10}$	0.011657714843750	$j_{2,42}$	0.005676269531250
$j_{2,11}$	0.011352539062500	$j_{2,43}$	0.005554199218750
$j_{2,12}$	0.011047363281250	$j_{2,44}$	0.005432128906250
$j_{2,13}$	0.010742187500000	$j_{2,45}$	0.005371093750000
$j_{2,14}$	0.010498046875000	$j_{2,46}$	0.005249023437500
$j_{2,15}$	0.010253906250000	$j_{2,47}$	0.005187988281250
$j_{2,16}$	0.010009765625000	$j_{2,48}$	0.005065917968750
$j_{2,17}$	0.009704589843750	$j_{2,49}$	0.005004882812500
$j_{2,18}$	0.009460449218750	$j_{2,50}$	0.004882812500000
$j_{2,19}$	0.009216308593750	$j_{2,51}$	0.004821777343750
$j_{2,20}$	0.009033203125000	$j_{2,52}$	0.004699707031250
$j_{2,21}$	0.008850097656250	$j_{2,53}$	0.004638671875000
$j_{2,22}$	0.008605957031250	$j_{2,54}$	0.004577636718750
$j_{2,23}$	0.008422851562500	$j_{2,55}$	0.004516601562500
$j_{2,24}$	0.008239746093750	$j_{2,56}$	0.004394531250000
$j_{2,25}$	0.008056640625000	$j_{2,57}$	0.004333496093750
$j_{2,26}$	0.007873535156250	$j_{2,58}$	0.004272460937500
$j_{2,27}$	0.007690429687500	$j_{2,59}$	0.004211425781250
$j_{2,28}$	0.007507324218750	$j_{2,60}$	0.004150390625000
$j_{2,29}$	0.007385253906250	$j_{2,61}$	0.004089355468750
$j_{2,30}$	0.007202148437500	$j_{2,62}$	0.004028320312500
$j_{2,31}$	0.007080078125000	$j_{2,63}$	0.003967285156250

Table A.9 The coefficients $c_{2,i}$ of 64-interval structure.

Coefficient	Value	Coefficient	Value
$c_{2,0}$	0.000183105468750	$c_{2,32}$	0.000061035156250
$c_{2,1}$	0.000183105468750	$c_{2,33}$	0.000061035156250
$c_{2,2}$	0.000183105468750	$c_{2,34}$	0.000061035156250
$c_{2,3}$	0.000183105468750	$c_{2,35}$	0.000061035156250
$c_{2,4}$	0.000183105468750	$c_{2,36}$	0.000061035156250
$c_{2,5}$	0.000183105468750	$c_{2,37}$	0.000061035156250
$c_{2,6}$	0.000122070312500	$c_{2,38}$	0.000000000000000
$c_{2,7}$	0.000122070312500	$c_{2,39}$	0.000000000000000
$c_{2,8}$	0.000122070312500	$c_{2,40}$	0.000000000000000
$c_{2,9}$	0.000122070312500	$c_{2,41}$	0.000000000000000
$c_{2,10}$	0.000122070312500	$c_{2,42}$	0.000000000000000
$c_{2,11}$	0.000122070312500	$c_{2,43}$	0.000000000000000
$c_{2,12}$	0.000122070312500	$c_{2,44}$	0.000000000000000
$c_{2,13}$	0.000122070312500	$c_{2,45}$	0.000000000000000
$c_{2,14}$	0.000122070312500	$c_{2,46}$	0.000000000000000
$c_{2,15}$	0.000122070312500	$c_{2,47}$	0.000000000000000
$c_{2,16}$	0.000122070312500	$c_{2,48}$	0.000000000000000
$c_{2,17}$	0.000061035156250	$c_{2,49}$	0.000000000000000
$c_{2,18}$	0.000061035156250	$c_{2,50}$	0.000000000000000
$c_{2,19}$	0.000061035156250	$c_{2,51}$	0.000000000000000
$c_{2,20}$	0.000061035156250	$c_{2,52}$	0.000000000000000
$c_{2,21}$	0.000061035156250	$c_{2,53}$	0.000000000000000
$c_{2,22}$	0.000061035156250	$c_{2,54}$	0.000000000000000
$c_{2,23}$	0.000061035156250	$c_{2,55}$	0.000000000000000
$c_{2,24}$	0.000061035156250	$c_{2,56}$	0.000000000000000
$c_{2,25}$	0.000061035156250	$c_{2,57}$	0.000000000000000
$c_{2,26}$	0.000061035156250	$c_{2,58}$	0.000000000000000
$c_{2,27}$	0.000061035156250	$c_{2,59}$	0.000000000000000
$c_{2,28}$	0.000061035156250	$c_{2,60}$	0.000000000000000
$c_{2,29}$	0.000061035156250	$c_{2,61}$	0.000000000000000
$c_{2,30}$	0.000061035156250	$c_{2,62}$	0.000000000000000
$c_{2,31}$	0.000061035156250	$c_{2,63}$	0.000000000000000

A.2 The coefficients in LUTs of NR Iteration

Table A.10 The initial guess values of one-stage NR iteration.

Initial guess	Value	Initial guess	Value	Initial guess	Value
1	0.998046875	39	0.869140625	77	0.771484375
2	0.994140625	40	0.865234375	78	0.767578125
3	0.990234375	41	0.865234375	79	0.765625
4	0.986328125	42	0.859375	80	0.76171875
5	0.982421875	43	0.857421875	81	0.76171875
6	0.978515625	44	0.853515625	82	0.7578125
7	0.974609375	45	0.853515625	83	0.755859375
8	0.970703125	46	0.84765625	84	0.755859375
9	0.966796875	47	0.84765625	85	0.751953125
10	0.962890625	48	0.841796875	86	0.748046875
11	0.958984375	49	0.841796875	87	0.74609375
12	0.958984375	50	0.837890625	88	0.74609375
13	0.953125	51	0.833984375	89	0.7421875
14	0.94921875	52	0.833984375	90	0.7421875
15	0.947265625	53	0.830078125	91	0.73828125
16	0.943359375	54	0.826171875	92	0.736328125
17	0.939453125	55	0.826171875	93	0.734375
18	0.9375	56	0.8203125	94	0.732421875
19	0.93359375	57	0.8203125	95	0.732421875
20	0.927734375	58	0.814453125	96	0.728515625
21	0.927734375	59	0.814453125	97	0.724609375
22	0.923828125	60	0.8125	98	0.72265625
23	0.91796875	61	0.80859375	99	0.72265625
24	0.916015625	62	0.806640625	100	0.71875
25	0.912109375	63	0.802734375	101	0.71875
26	0.908203125	64	0.802734375	102	0.71484375
27	0.904296875	65	0.798828125	103	0.71484375
28	0.904296875	66	0.794921875	104	0.7109375
29	0.900390625	67	0.79296875	105	0.7109375
30	0.896484375	68	0.79296875	106	0.70703125
31	0.89453125	69	0.787109375	107	0.70703125
32	0.888671875	70	0.787109375	108	0.703125
33	0.888671875	71	0.78515625	109	0.703125
34	0.884765625	72	0.78125	110	0.69921875
35	0.880859375	73	0.779296875	111	0.69921875
36	0.876953125	74	0.775390625	112	0.6953125
37	0.876953125	75	0.775390625	113	0.6953125

38	0.87109375	76	0.771484375	114	0.69140625
115	0.69140625	158	0.62109375	201	0.560546875
116	0.689453125	159	0.619140625	202	0.55859375
117	0.689453125	160	0.6171875	203	0.556640625
118	0.685546875	161	0.6171875	204	0.556640625
119	0.681640625	162	0.61328125	205	0.5546875
120	0.681640625	163	0.61328125	206	0.5546875
121	0.6796875	164	0.61328125	207	0.5546875
122	0.6796875	165	0.609375	208	0.55078125
123	0.67578125	166	0.609375	209	0.55078125
124	0.67578125	167	0.607421875	210	0.55078125
125	0.671875	168	0.60546875	211	0.548828125
126	0.671875	169	0.60546875	212	0.546875
127	0.66796875	170	0.6015625	213	0.546875
128	0.66796875	171	0.6015625	214	0.546875
129	0.666015625	172	0.6015625	215	0.54296875
130	0.6640625	173	0.59765625	216	0.54296875
131	0.6640625	174	0.59765625	217	0.541015625
132	0.66015625	175	0.595703125	218	0.541015625
133	0.66015625	176	0.59375	219	0.5390625
134	0.65625	177	0.59375	220	0.5390625
135	0.65625	178	0.591796875	221	0.537109375
136	0.65234375	179	0.58984375	222	0.537109375
137	0.65234375	180	0.58984375	223	0.533203125
138	0.650390625	181	0.587890625	224	0.533203125
139	0.6484375	182	0.5859375	225	0.53125
140	0.6484375	183	0.5859375	226	0.53125
141	0.64453125	184	0.583984375	227	0.53125
142	0.64453125	185	0.58203125	228	0.529296875
143	0.642578125	186	0.58203125	229	0.529296875
144	0.640625	187	0.578125	230	0.529296875
145	0.640625	188	0.578125	231	0.52734375
146	0.63671875	189	0.578125	232	0.525390625
147	0.63671875	190	0.57421875	233	0.5234375
148	0.6328125	191	0.57421875	234	0.521484375
149	0.6328125	192	0.57421875	235	0.521484375
150	0.6328125	193	0.572265625	236	0.51953125
151	0.62890625	194	0.572265625	237	0.51953125
152	0.62890625	195	0.568359375	238	0.51953125
153	0.626953125	196	0.568359375	239	0.517578125
154	0.625	197	0.56640625	240	0.515625
155	0.625	198	0.564453125	241	0.515625
156	0.625	199	0.564453125	242	0.515625

157	0.62109375	200	0.560546875	243	0.513671875
244	0.51171875	249	0.5078125	254	0.501953125
245	0.509765625	250	0.5078125	255	0.501953125
246	0.509765625	251	0.50390625	256	0.501953125
247	0.509765625	252	0.50390625		
248	0.5078125	253	0.501953125		

Table A.11 The initial guess values of two-stage NR iteration.

Initial guess	Value	Initial guess	Value
1	0.96875	9	0.626953125
2	0.91796875	10	0.626953125
3	0.87890625	11	0.580078125
4	0.83984375	12	0.580078125
5	0.779296875	13	0.580078125
6	0.763671875	14	0.5234375
7	0.69921875	15	0.5234375
8	0.69921875	16	0.5234375

A.3 The power consumptions at various frequencies.

Table A.12 The total power consumptions at various frequencies.

Frequency(Hz) /Power(W)	16HPS	32HPS	64HPS	1NR	2NR
1	1.840e-08	1.684e-08	1.681e-08	1.463e-08	1.906e-08
10	1.853e-08	1.697e-08	1.691e-08	1.475e-08	1.932e-08
100	1.980e-08	1.821e-08	1.789e-08	1.595e-08	2.193e-08
1000	3.247e-08	3.068e-08	2.770e-08	2.792e-08	4.803e-08
10000	1.592e-07	1.553e-07	1.258e-07	1.484e-07	3.099e-07
100000	1.427e-06	1.402e-06	1.106e-06	1.366e-06	2.918e-06
1000000	1.410e-05	1.387e-05	1.091e-05	1.346e-05	2.902e-05
10000000	1.409e-04	1.385e-04	1.090e-04	1.346e-04	2.901e-04
100000000	1.661e-03	1.436e-03	1.090e-03	1.363e-03	4.977e-03

Table A.13 The switching power, internal power, static power and total power for 64-interval architecture after post synthesis simulation

Frequency(Hz) /Power(W)	Switching Power	Internal Power	Static Power	Total Power
1	5.735e-12	5.169e-12	1.680e-08	1.681e-08
10	5.735e-11	5.169e-11	1.680e-08	1.691e-08
100	5.735e-10	5.169e-10	1.680e-08	1.789e-08
1000	5.733e-09	5.164e-09	1.680e-08	2.770e-08
10000	5.733e-08	5.164e-08	1.680e-08	1.258e-07
100000	5.733e-07	5.164e-07	1.680e-08	1.106e-06
1000000	5.733e-06	5.164e-06	1.680e-08	1.091e-05
10000000	5.733e-05	5.164e-05	1.680e-08	1.090e-04
100000000	5.733e-04	5.166e-04	1.680e-08	1.090e-03

Table A.14 The switching power, internal power, static power and total power for 1NR architecture after post synthesis simulation

Frequency(Hz) /Power(W)	Switching Power	Internal Power	Static Power	Total Power
1	7.188e-12	6.140e-12	1.462e-08	1.463e-08
10	7.188e-11	6.140e-11	1.462e-08	1.475e-08
100	7.188e-10	6.140e-10	1.462e-08	1.595e-08
1000	7.166e-09	6.121e-09	1.463e-08	2.792e-08
10000	7.205e-08	6.168e-08	1.463e-08	1.484e-07
100000	7.275e-07	6.237e-07	1.462e-08	1.366e-06
1000000	7.233e-06	6.209e-06	1.464e-08	1.346e-05
10000000	7.241e-05	6.216e-05	1.465e-08	1.346e-04
100000000	7.355e-04	6.273e-04	1.462e-08	1.363e-03

Table A.15 The switching power, internal power, static power and total power for 64-interval architecture after post layout simulation

Frequency(Hz) /Power(W)	Switching Power	Internal Power	Static Power	Total Power
1	1.182e-11	1.106e-11	1.714e-08	1.717e-08
10	1.182e-10	1.106e-10	1.714e-08	1.737e-08
100	1.171e-09	1.096e-09	1.705e-08	1.932e-08
1000	1.178e-08	1.103e-08	1.712e-08	3.992e-08
10000	1.185e-07	1.108e-07	1.711e-08	2.464e-07
100000	1.174e-06	1.098e-06	1.710e-08	2.290e-06
1000000	1.175e-05	1.100e-05	1.699e-08	2.276e-05
10000000	1.180e-04	1.104e-04	1.700e-08	2.284e-04
100000000	1.225e-03	1.150e-03	1.765e-08	2.376e-03

Table A.16 The switching power, internal power, static power, total power for 1NR architecture after post layout simulation

Frequency(Hz) /Power(W)	Switching Power	Internal Power	Static Power	Total Power
1	1.293e-11	1.104e-11	1.476e-08	1.479e-08
10	1.293e-10	1.104e-10	1.476e-08	1.500e-08
100	1.293e-09	1.104e-09	1.476e-08	1.716e-08
1000	1.296e-08	1.104e-08	1.475e-08	3.875e-08
10000	1.322e-07	1.120e-07	1.473e-08	2.589e-07
100000	1.335e-06	1.131e-06	1.470e-08	2.480e-06
1000000	1.318e-05	1.114e-05	1.476e-08	2.434e-05
10000000	1.350e-04	1.145e-04	1.472e-08	2.496e-04
100000000	1.359e-03	1.180e-03	1.506e-08	2.539e-03

References

[1] E. Hertz, "*Parabolic Synthesis*," Series of Licentiate and Doctoral Thesis at the Department of Electrical and Information Technology, Faculty of Engineering, Lund University, Sweden, January 2011, *ISBN 9789174730692*, http://www.eit.lth.se/fileadmin/eit/news/747/lic_14.5_mac.pdf

[2] P.K. Meher, J. Valls, T.B. Juang, K. Sridharan, and K. Maharatna, "50 years of CORDIC: Algorithms, architectures, and applications," *IEEE Transactions on Circuits & Systems. Part I: Regular Papers*, Vol. 56, Issue 9, pp. 1893-1907, *ISSN 15498328*, September 2009.

[3] E. Hertz and P. Nilsson, "A Methodology for Parabolic Synthesis," a book chapter in *VLSI*, *In-Tech*, *ISBN 9783902613509*, pp. 199-220, Vienna, Austria, February 2010.

[4] D. Zuras, M. Cowlishaw, and others, "IEEE standard for floating-point arithmetic," IEEE Std 754-2008, pp. 1–70, doi: 10.1109/IEEESTD.2008.4610935, IEEE, 2008.

[5] E. Hertz, "Floating-point representation," Private communication, January 2015.

[6] 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 the proceedings of 20th European Conference on Circuit Theory & Design* (*ECCTD*), *ISBN 9781457706172*, pp. 709-712, Konsert & Kongress, Linkoping, Sweden, August 2011.

[7] E. Hertz and P. Nilsson, "Parabolic synthesis methodology implemented on the sine function," *in the proceedings of IEEE International Symposium on Circuits and Systems*, *ISBN 9781424438273*, pp. 253-256, Taipei, Taiwan, May 2009.

[8] K.S. Johnson, "*Transmission Circuits for Telephonic Communication: Methods of Analysis and Design*," D. Van Nostrand company, New York, USA, 1947.

[9] J. Lai, "*Hardware Implementation of the Logarithm Function using Improved Parabolic Synthesis*," Master of Science Thesis at the Department of Electrical and Information Technology, Faculty of Engineering, Lund University, Sweden, September 2013, http://www.eit.lth.se/sprapport.php?uid=751.

[10] P. Kornerup and J.M. Muller, "Choosing starting values for Newton-Raphson computation of reciprocals, square roots and square root reciprocals," *in the proceedings of 5th conference on Real Numbers and Computers*, Leon, France, September 2003.

[11] P. Montuschi and M. Mezzalama, "Optimal absolute error starting values for Newton–Raphson calculation of square root," *Computing*, Vol. 46, Issue 1, pp. 67-86, *ISSN 0010485X*, Springer-Verlag, 1991.

[12] M.J. Schulte, J. Omar, and E.E. Swartzlander, "Optimal initial approximation for the Newton–Raphson division algorithm," *Computing*, Vol. 53, Issue. 3/4, pp. 233-242, *ISSN 0010485X*, Springer-Verlag, 1994.

[13] E. Hertz and P. Nilsson, "A methodology for parabolic synthesis of unary functions for hardware implementation," *in the proceedings of the 2008 International conference on Signals, Circuits & Systems* (*SCS08*), *ISBN 9781424426270*, pp. 30-35, Hammamet, Tunisia, November 2008.

[14] A.A. Giunta and L.T. Watson, "A Comparison of Approximation Modeling Techniques: Polynomial Versus Interpolating Models," *in the proceedings of 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Multidisciplinary Analysis Optimization Conferences*, Patent No. AIAA-98-4758, pp. 392-404, American Institute of Aeronautics and Astronautics, Blacsburg, USA, September 1998.

[15] A. Agarwal, S. Mukhopadhyay, A. Raychowdhury, K. Roy, and C.H. Kim, "Leakage power analysis and reduction for nanoscale circuits," *IEEE Micro*, Vol. 26, Issue 2, pp. 68-80, *ISSN 02721732*, October 2006.

[16] G. Yip, "Expanding the Synopsys Prime Time Solution with Power Analysis," Synopsys, Inc., http://www.synopsys.com/, June 2006.

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2016-487

http://www.eit.lth.se