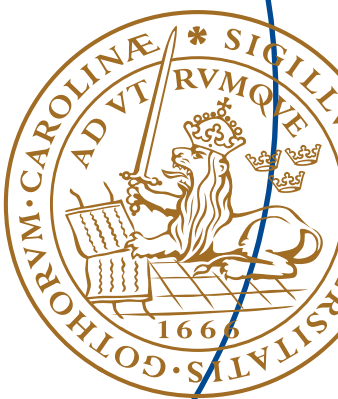
Master's Thesis

Parabolic Synthesis and Non-Linear Interpolation

Adeel Muhammad Hashmi



Department of Electrical and Information Technology,
Faculty of Engineering, LTH, Lund University, January 2015.

Master's Thesis

Parabolic Synthesis and Non-Linear Interpolation

By

Adeel Muhammad Hashmi

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

Abstract

Computation and implementation of unary functions such as trigonometric, logarithmic and exponential function have a vital importance in modern applications, e.g., Digital Signal Processing, computer graphics, wireless systems and virtual reality simulations. Over the past few years many software solutions have been used, which provide extreme precision but take a lot of computation time for real-time applications. As compared to the software routines, a hardware implementation of unary function is found to be a best solution for real-time applications where fast and numerically intensive solutions are required.

This thesis work presents an approximation of trigonometric functions, i.e. Sine and Cosine using Parabolic Synthesis combined with Non-Linear Interpolation. The architecture for the approximation is designed and implemented in the stm65 CMOS technology. There is a high degree of parallelism in the design which makes it faster than other methodologies to calculate unary functions. The same design can be used to implement various kinds of unary function like logarithmic and exponential etc. with the same architecture.

The design is compared, with respect to power consumption, area and maximum speed, with the existing methodologies like the CORDIC, Parabolic Synthesis, and the Parabolic Synthesis with Linear Interpolation. It is found that the architecture has better performance in terms of chip area, speed and power consumption.

Acknowledgments

I would like to begin to express my sincere thanks and gratitude towards ***Prof. Peter Nilsson*** and ***Erik Hertz***, Supervisor, to provide me with this opportunity to experiences this research oriented MS Thesis entitled: "***Parabolic Synthesis & Non-Linear Interpolation***", in the field of Digital ASIC. I would appreciate their explicit guidance, prolific command and remarkable knowledge about efficient algorithms for computation and implementation of unary functions using innovative techniques. Without their continuous help and guidance this thesis work would not have been possible.

Next, I would like to cordially thank Pia Bruhn, Program Coordinator EIT Department, Lund University, for being a guardian and helping me during my studies tenure. She was really marvelous to tackle problems and difficulties that a newcomer faces when they arrive in a new country; guided and helped me out with the administrative issues in the best possible manner whenever I requested for intervention and assistance. My thanks and warm regards also goes to Anna Carlqvist and Helene Von Wachenfelt, the International Master's Coordinators, for their help and guidance in study administration and residence permit issues.

Then I wish to continue by thanking Dr. S.M. Yasser Sherazi and Dr. Taimoor Abbas for providing me there sincere guidance of how to tackle problems step-by-step and move forwards towards progressive development. Also these two people have

been a really good resource when it came to technical discussions and knowledge sharing.

Now I will concentrate my attention to express my gratitude towards my friends and colleagues here at Lund University, without whom the time spent here in Sweden would not have been joyful. I would appreciate whole heartedly my friends Waqas Shafiq and Karrar Rizvi to help me out in the basic understanding and developing my competencies and skill set and giving me a push to complete this Thesis Work, while being part of this Master's Degree. I would like to thank Shabraiz Muhammad for proof reading my thesis and guiding me with Technical Report Writing Skills. I would gladly express my gratitude towards colleagues Shoaib, Adnan, Naveed, Azhar, Farhan, Sardar Sulaman, Aadil and Rizwan for the group activities, bar-be-cues and evening gatherings. I would also like to thank Jovita, Minna, Erica and Justyna for their love, care and affection during my stay in Sweden and so on.

Last but the most important of all, I would like to thank my family, especially my mother for all her love, care, affection, support and prayers.

Adeel Muhammad Hashmi

Table of Contents

Chapter 1

1 Introduction

With the advent of Chip Technology the size of technical equipments and electronics hardware have reduced significantly. This is being perceived as the future of Next Generation Technologies. In olden days, canon sized devices were used for complex computations and calculations. Nowadays digital circuits and devices of mere existence possessing the ability to perform similar objectives by utilizing these limited resources namely: memory, time of execution and power.

We can observe in our surroundings that there is increase in demand for ultra-low weight, less power consuming and super-efficient devices over the past few years. General public is unaware of the challenges faced by the researchers in order to attain these said objectives. The researchers try to make ends meet by working to devise ways and methods to produce equipments that can provide the optimum performance with effective utilization of the aforementioned limited resources. This Master's Thesis comprises of a study and comparative analysis conducted to ensure usage of the Parabolic Synthesis and Non-Linear Interpolation. It also provides the knowledge about how this next generation computational methodology can be fruitful, if their architectures are implemented in real time systems.

Computation and implementation of unary functions such as trigonometric, logarithmic and exponential function have a vital importance in modern applications, e.g., Digital Signal Processing (DSP), computer graphics (2D/3D), wireless systems and virtual reality simulations. Over the past few

years many software solutions have been used, which provide extreme precision but take a lot of computation time for real-time applications. As compared to the software routines, a hardware implementation of unary function is found to be a best solution for real-time applications where fast and numerically intensive solutions are required.

There are different methods that are employed for hardware implementation of unary functions. The easiest method is by using look-up table [1] [2]. It is an efficient method for low precision computations where the input word-length is between 12-16 bits which corresponds to a table size of 4096-65536 words.

$$Table\ size\ =\ 2^n \tag{1.1}$$

Where *n* is the input word-length.

It can be seen in (1.1) that the table size will increase exponentially with the increased number of input word-length. Therefore for high precision applications the execution time will be large and unacceptable in certain cases.

With the evolution of the various industrial sectors like DSP, Robotics, Communication Systems, there has been an increase in demand of high speed hardware implementations. A variety of solutions have been proposed ranging from implementation of algorithms that utilize the lookup tables for low precision computations [9]. Various other hardware approaches have been implemented e.g. CORDIC [9] & Polynomial based approximation e.g. Taylor Series Implementation [9] [14].

Polynomial based approximation is another method that is being used for computing the unary functions. It has an advantage of being table-less but it introduces large number of computational complexities since it is performed with multipliers and adders. The computational complexity of

this method can be reduced by combining it with look-up table methods. Taylor polynomial is an example of such scheme [3]. Designing an efficient approximation for the function to be approximated is the key in polynomial based approximations [4].

COordinate Rotation DIgital Computer (CORDIC) is a widely used algorithm for hardware implementation of basic elementary function like logarithmic, trigonometric, exponential etc. It was proposed by Jack E.Volder in 1959 to provide the real-time digital solution for navigational computations [5] [6]. It is an iterative method that requires simple shift and add operation together with a small look-up table [7]. Therefore it is used in designs where different design aspects like critical speed, low area and low power consumption are of vital importance. Since it is an iterative method, it produces one extra bit of accuracy in each rotation [8]. For higher accuracy applications, CORDIC method will require more iterations in order to get better resolution. That will increase the execution time of the operation therefore it will be insufficient for very high speed applications.

A new methodology Parabolic Synthesis has recently been proposed by Erik Hertz and Peter Nilsson to perform the realization of unary functions like trigonometric, logarithms as well as division and square-root functions in hardware [9] [10]. The parallel architecture of this method increases the performance and decrease the power, area and speed limitations compared to previously mentioned algorithms including CORDIC. The main feature of parabolic architecture is that it can be used for the realization of different unary functions. Only the coefficients need to be changed for different functions but the hardware will remain fixed. Thus the design will remain the same and can be directly used without any changes for other applications [8].

In this thesis, a methodology is presented by combining parabolic synthesis with non-linear interpolation for the realization of trigonometric functions sine and cosine. Parabolic methodology is a synthesis of second order functions which provides accuracy depending on the number of second order functions [7]. In the combined methodology, the accuracy depends on

the number of intervals in the non-linear interpolation. Furthermore, the behavior and optimization of coefficients for the implementation of trigonometric functions, sine and cosine, is discussed.

The proposed architecture is designed using two stages of parabolic synthesis [11] where the second stage is implemented as a non-linear interpolation in the stm65 CMOS technology. The design is simulated and compared for accuracy, power consumption and performance. The core area is also estimated. Synthesized VHDL is used in the project. Low Power High VT and Low Power Low VT transistors are used, in separate designs. Three different supply voltages, *VDD* = {1.00, 1.10, 1.20} volts are used. The power and energy consumption, both static and dynamic, are estimated. The design is compared, with respect to power consumption, area and maximum speed, with the existing methodologies like CORDIC, Parabolic Synthesis, and Parabolic Synthesis with Linear Interpolation.

Chapter 2

2 Parabolic Synthesis and Non-Linear Interpolation

2.1 Parabolic Synthesis

The Parabolic Synthesis Methodology is a Hardware Approach proposed by Erik Hertz and Peter Nilsson in order to develop functions to perform approximation in the hardware [9]. The implementation involves a parallel architecture for providing solution to the complex computational problem to reduce execution time. In parabolic synthesis methodology an approximation of unary functions in hardware is dealt with.

This methodology is based on second order parabolic functions, called sub-functions $s_n(x)$ [7]. These sub-functions are multiplied together to found the original function $f_{org}(x)$ as shown in (2.1) [14]. The original function is the product of all sub-functions, when the number of sub-functions approaches infinity. The sub-function must satisfy that the function is limited to the range $0 \leq x < 1$ and $0 \leq f(x) < 1$.

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

In order to gradually develop sub-functions, we need to determine the first help function. First help function is the ratio of original function and first sub-function, i.e. $s_1(x)$.

$$f_1(x) = \frac{forg(x)}{s_1(x)} = s_2(x) \,.\, s_3(x) \cdots . s_\infty(x) \tag{2.2}$$

The individual help functions can be generalized to be evaluated as:

$$f_n(x) = \frac{f_{n-1}(x)}{s_n(x)} = s_{n+1}(x) \,.\, s_{n+2}(x) \,.\ldots.\, . s_\infty(x) \tag{2.3}$$

These help functions are in turn used to compute the values of sub functions by performing normalization. These sub-functions are constructed as second or polynomials depicting the parabolic functions [14].

2.1.1 Normalization

First the function to be approximated has to be normalized according to the parabolic synthesis methodology. Normalization limits the function in a numerical range to facilitate the hardware implementation. It must satisfy that the function is limited to the range $0 \leq x < 1$ and $0 \leq f(x) < 1$. Starting and ending coordinate should be (0,0) and less than (1,1) respectively [14].

2.1.2 First sub-function

In order to develop the first sub-function, $s_1(x)$, the original function, $f_{org}(x)$, should cross two points i.e., (0,0) and (1,1) as shown in the Fig. 2.1.

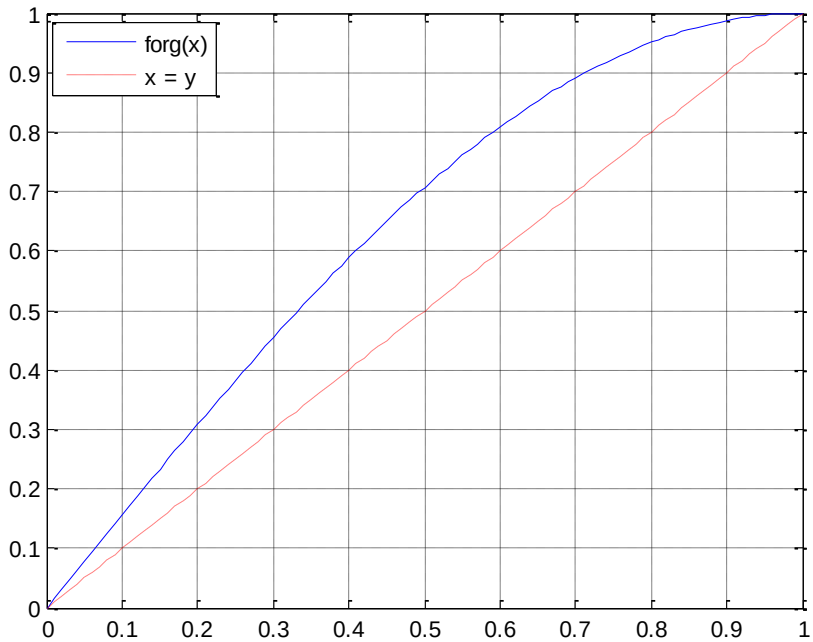


Figure 2.1: Comparison of original function, $f_{org}(x)$, with straight line x=y

The first sub-function, $s_1(x)$, is a second order parabolic function as define by the (2.4).

$$s_1(x) = l_1 + k_1 x + c_1(x - x^2) \tag{2.4}$$

The starting point, l_1, of first sub-function, $s_1(x)$, is calculated to be zero as it crosses (0,0). As the function lies between the points, (0,0) and (1,1), the slope k_1 is 1 [7] [9] [16]. Therefore, the first sub-function can be simplified as shown in (2.5).

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

The coefficient c_1 is computed according to (2.6).

$$c_1 = \lim_{x \to 0} \frac{f_{org}(x)}{x} - 1 \tag{2.6}$$

2.1.3 Second sub-function

In order to make the total error smaller, the second sub-function,$s_2(x)$, is developed to approximate the value of first help function, $f_1(x)$. A strictly convex or concave first help function, $f_1(x)$, can be developed from original function, $f_{org}(x)$, using (2.2) [16].

The second sub-function, $s_2(x)$, can be defined as shown in (2.7).

$$s_2(x) = l_2 + k_2 x + c_2(x - x^2) \tag{2.7}$$

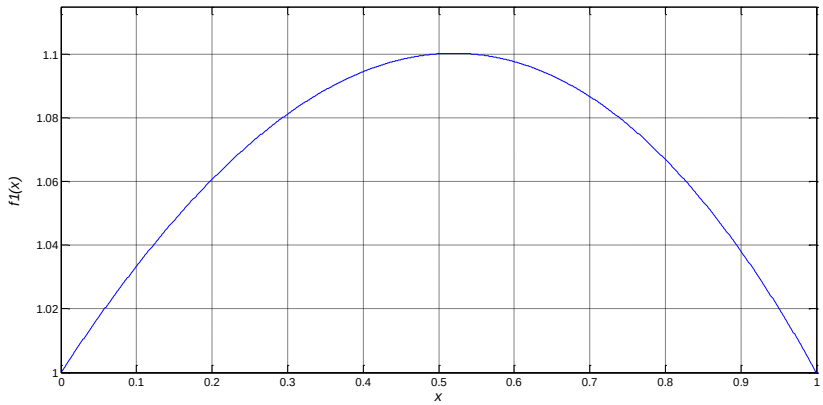


Figure 2.2: A strictly convex first help function, $f_1(x)$.

As it can be seen in Fig. 2.2 that the second sub-function starts at a point (0,1) and finishes at (1,1), so the starting point, l_2, of second sub-function is 1 and the slope, k_2, of the function is 0. Therefore the equation for second sub-function can be reduced as shown in (2.8).

$$s_2(x) = 1 + c_2(x - x^2) \tag{2.8}$$

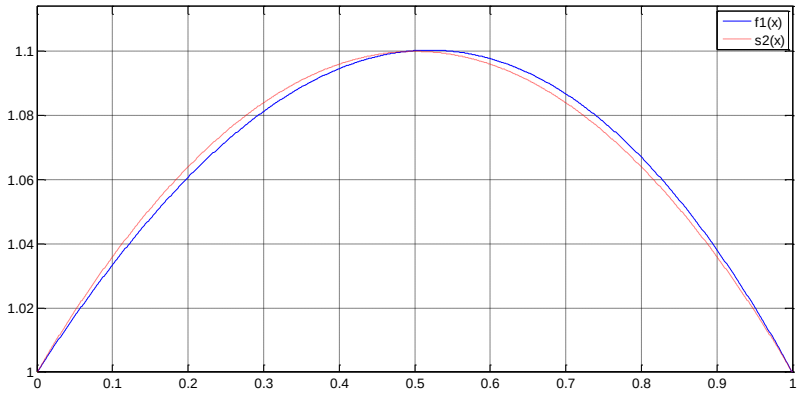


Figure 2.3: Comparison of first help function, $f_1(x)$, with second sub-function, $s_2(x)$.

In order to develop and verify second sub-function, it must cross the starting point, middle point and the end point of the help function as shown in Fig. 2.3.

2.1.4 Sub-functions for $n > 2$

In order to develop further sub-functions, $s_n(x)$ for $n > 2$, same methodology is applied as given in (2.2) and (2.3). However, the functions will not be strictly convex or concave in the range of 0 to 1. For example, the function, $f_2(x)$, shown in fig. 2.4 is a pair of convex and concave functions. The first function is in the range $\{0 \leq x < 0.5\}$ and the second function is in the range $\{0.5 \leq x < 1.0\}$.

Therefore the second help function can be expressed as (2.9).

$$f_2(x) = \begin{cases} f_{2,0}(x) & 0 \leq x < \frac{1}{2} \\ f_{2,1}(x) & \frac{1}{2} \leq x < 1 \end{cases} \tag{2.9}$$

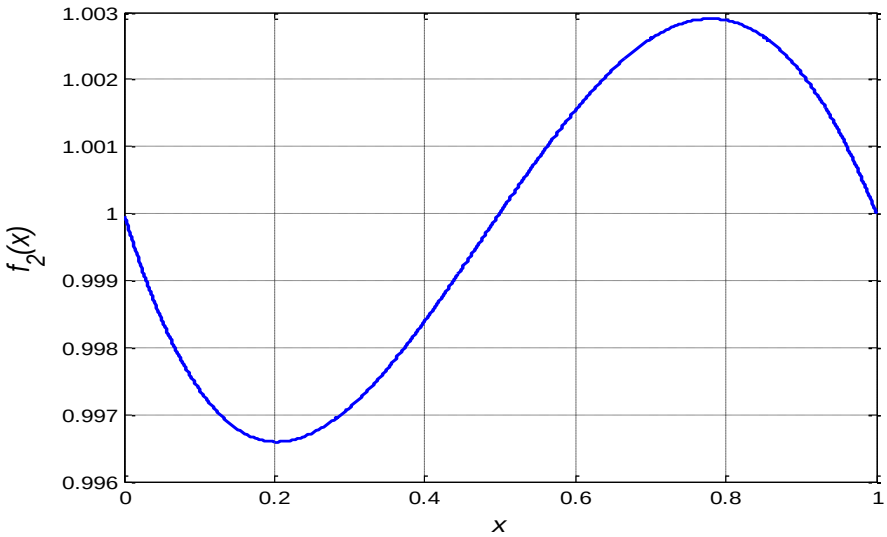


Figure 2.4: Second help function, $f_2(x)$, pair of opposite convex and concave functions.

The approximation of a function which is composed of two parabolic curves can be performed by normalizing each curve in the interval 0 to 1 on x axis. In order to map the input x to the normalized parabolic curve, *x* can be replaced with *x*' as shown in (2.10).

$$x' = frac(2.x) \tag{2.10}$$

The approximation of each parabolic curve is performed as described in Section 2.1.3. In order to approximate the third sub-function, $s_{3,0}(x')$ is calculated when $0 \leq x < \frac{1}{2}$ and $s_{3,1}(x')$ is calculated when $\frac{1}{2} \leq x < 1$ as given in (2.11).

$$s_3(x) = \begin{cases} s_{3,0}(x') & 0 \leq x < \frac{1}{2} \\ s_{3,1}(x') & \frac{1}{2} \leq x < 1 \end{cases} \tag{2.11}$$

A larger number of n results in higher number of convex and concave functions. The methodology can be generalized to calculate the n^{th} help function as shown in (2.12).

$$f_{\mathrm{n}}(x) = \begin{cases} f_{n,0}(x), & 0 \le x < \frac{1}{2^{n-1}} \\ f_{n,1}(x), & \frac{1}{2^{n-1}} \le x < \frac{2}{2^{n-1}} \\ \quad\quad ... & \quad\quad ... \\ f_{n,2^{n-1}-1}(x), & \frac{2^{n-1}-1}{2^{n-1}} \le x < 1 \end{cases} \tag{2.12}$$

Using these partial help functions, the corresponding sub-function are developed. The sub-function is also divided into partial sub-functions as given in (2.13).

$$s_{\mathrm{n}}(x) = \begin{cases} s_{n,0}(x), & 0 \le x < \frac{1}{2^{n-2}} \\ s_{n,1}(x), & \frac{1}{2^{n-2}} \le x < \frac{2}{2^{n-1}} \\ \quad\quad ... & \quad\quad ... \\ s_{n,2^{n-1}-2}(x), & \frac{2^{n-2}-1}{2^{n-2}} \le x < 1 \end{cases} \tag{2.13}$$

In the same way, the input x is substituted by x_n to map the input to the normalized parabolic curve.

$$x_n = frac(2^{n-2}.x) \tag{2.14}$$

Similar to the second sub-function given in (2.8), the start value of each of the partial help function is 1 and the end value of each partial help function, $f_{\mathrm{n-1,m}}(x)$, interval is also 1. Therefore, the gradient, $k_{\mathrm{n,m}}$, of each sub-function is 0. This enables to reduce the sub-function as shown in (2.15).

$$\begin{aligned} s_{n,m}(x_n) &= l_{n,m} + k_{n,m}x_n + c_{n,m}(x_n - x_n^2) \\ &= 1 + c_{n,m}(x_n - x_n^2) \end{aligned} \tag{2.15}$$

The coefficients, $c_{n,m}$, are calculated in such a way to satisfy the quotient between help function, $f_{n-1,m}(x)$, and the partial sub-function, $s_{n,m}(x)$, is equal to 1, when x_n is equal to 0.5.

$$c_{n,m} = 4 \times \left(f_{n-1,m} \left(\frac{2.(m+1)-1}{2^{n-1}} \right) - 1 \right) \quad (2.16)$$

2.2 Interpolation

Interpolation is a method of finding new data points from a set a known data points.

2.2.1 Linear Interpolation

Linear interpolation is the simplest method of interpolation. It takes two data points to construct the value of new data points. The classical linear interpolation for two data points is shown in (2.17).

$$y = y_b(n) + \left((x_b(n+1) - x_b(n)) \left\{ \frac{y_b(n+1) - y_b(n)}{x_b(n+1) - x_b(n)} \right\} \right) \quad (2.17)$$

In (2.17), $x_b(n)$ is the starting and $x_b(n+1)$ is the ending breakpoint of each interval. $y_b(n)$ and $y_b(n+1)$ are the respective y value at these breakpoints [14]. Linear interpolation using two intervals is shown in Fig. 2.5. It can be seen that $x_b = \{0, 2^{-1}\}$ for first interval and $x_b = \{2^{-1}, 1\}$ for the second interval. Equation (2.18) shows the corresponding y_b values.

$$y_b = 0, \frac{1}{\sqrt{2}}, and\ 1 \quad (2.18)$$

More intervals can be used for better accuracy, e.g. four intervals, that give the breakpoint values as shown in (2.19). For the sake of hardware architecture, breakpoints are always the power of number 2 [14].

$$x_b = \{0, 2^{-2}, 2 \times 2^{-2}, 3 \times 2^{-2}, 1\} \quad (2.19)$$

For more intervals, equation (2.17) can be modified as shown in (2.20).

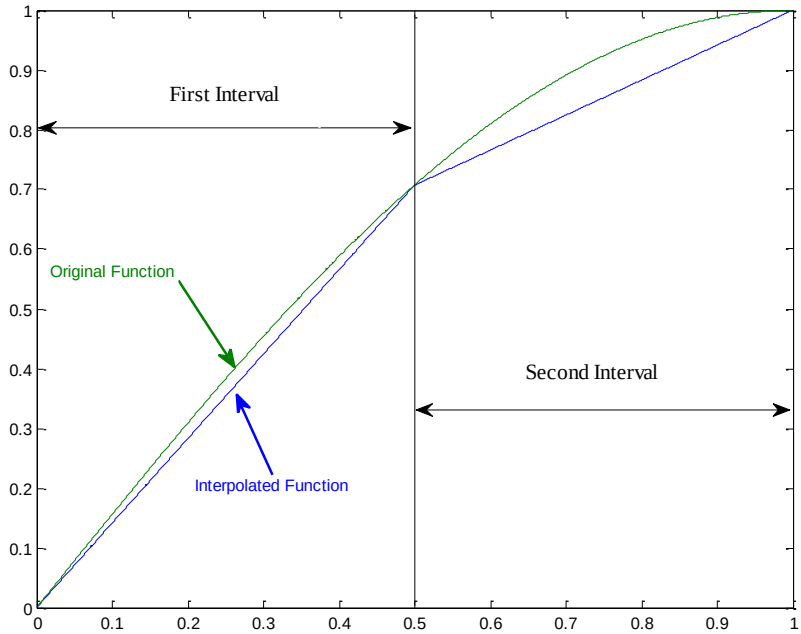


Figure 2.5: Linear interpolation of a normalized function

$$y = y_b(n) + \ v \ \times \left((x_b(n+1) - \ x_b(n)) \left\{ \frac{y_b(n+1) - y_b(n)}{v \times (x_b(n+1) - x_b(n))} \right\} \right) \quad (2.20)$$

Where v is the number of intervals. For example, for $v = 4$ we get (2.21)

$$y = y_b(n) + \ 4 \ \times \left((x_b(n+1) - \ x_b(n)) \left\{ \frac{y_b(n+1) - y_b(n)}{4 \times (x_b(n+1) - x_b(n))} \right\} \right) \quad (2.21)$$

Or (2.22)

$$y = y_b(n) + 4 \times \left((x_b(n+1) - x_b(n)) \left\{ \frac{y_b(n+1) - y_b(n)}{4 \times (0.25 - 0)} \right\} \right)$$

$$y = y_b(n) + 4 \times \left((x_b(n+1) - x_b(n)) \left\{ \frac{y_b(n+1) - y_b(n)}{4 \times (0.5 - 0.25)} \right\} \right)$$

(2.22)

$$y = y_b(n) + 4 \times \left((x_b(n+1) - x_b(n)) \left\{ \frac{y_b(n+1) - y_b(n)}{4 \times (0.75 - 0.5)} \right\} \right)$$

$$y = y_b(n) + 4 \times \left((x_b(n+1) - x_b(n)) \left\{ \frac{y_b(n+1) - y_b(n)}{4 \times (1.0 - 0.75)} \right\} \right)$$

A good property of (2.22) is that the denominator is always "1" as division is not suggested for a hardware design. It is appreciated for other more hardware reasons as well. For v intervals, the linear interpolation is shown in (2.23) [14].

$$y = y_b(n) + v \times \left(x_b(n+1) - x_b(n)\right) \times \left(y_b(n+1) - y_b(n)\right) \quad (2.23)$$

2.2.2 Non-linear Interpolation

This thesis work is about parabolic synthesis and non-linear interpolation. The non-linear interpolation follows the same idea as the linear interpolation, with the difference that the approximations in the intervals are parabolic functions [14].

$$s_2(x) = l_{2,i} + k_{2,i} \times x_w + c_{2,i} \times (x_w - x_w^2) \quad (2.24)$$

The second stage (2.24) is a non-linear interpolation of the first help function, where $2, i$ index stands for the intervals in which the interpolation is performed. The interval index i is a power of 2, which gives the number of intervals, i.e. 1, 2, 4, 8, 16 and so on. For instance, in the case of 2, there will be two intervals, the first $\{0 \leq x < 0.5\}$ and the second $\{0.5 \leq x < 1.0\}$ in a normalized space$\{0 \leq x < 1.0\}$.

The index w, in (2.24), shows that the x term in the interpolation stage is dependent in the number of intervals used in the interpolation. The x term is affected in such a way that when two intervals ($w = 1$) are used in the interpolation then the most significant bit is thrown away. When four intervals ($w = 2$) are used in the interpolation then the two most significant bits are thrown away in the x term. The second sub-function can be divided into two parts, a linear part shown in (2.25) and a non-linear part as shown in (2.26) [14].

$$l_{2,i} + k_{2,i} \times x_w \tag{2.25}$$

$$c_{2,i} \times (x_w - x_w^2) \tag{2.26}$$

In (2.25) there are two coefficients for interpolation in each interval, a starting point, $l_{2,i}$, and a gradient, $k_{2,i}$. The starting point of an interval for the interpolation can be calculated by placing the value of *x* for the starting point of the interval $x_{\text{start},i}$, to the first help function, $f_1(x)$ [14].

$$l_i = f_1(x_{\text{start},i}) \tag{2.27}$$

The second coefficient, k_i, is the gradient of the interval in which the interpolation is being performed. The gradient is calculated by subtracting the end point value, $f_1(x_{\text{end},i})$, from the start point value, $f_1(x_{\text{start},i})$, of the interval [14].

$$k_i = f_1(x_{\text{end},i}) - f_1(x_{\text{start},i}) \tag{2.28}$$

As it is mentioned before that the intervals are normalized, so there is no denominator needed.

In (2.26), $c_{2,i}$is calculated in advance so that the second sub-function, $s_2(x)$, for the corresponding interval cuts the first help function, $f_1(x)$, in the middle of the interval i. therefore it satisfies the middle point, $x_{middle,i}$, for $f_1(x)$, as shown in (2.29) [14].

$$c_{2,i} = 4 \times (f_1(x_{\text{middle},i}) - l_i - k_i \times 0.5) \tag{2.29}$$

In (2.30) we have a simplification of (2.24). This simplification reduces an adder in hardware implementation.

$$s_2(x) = l_{2,i} + j_{2,i} \,.\, x_w - c_{2,i} \,.\, x_w^2 \tag{2.30}$$

Where

$$j_{2,i} = k_{2,i} + c_{2,i} \tag{2.31}$$

2.3 Parabolic Synthesis Combined with Interpolation

The drawback with parabolic synthesis is that if we want to increase the accuracy of the approximated function, the number of sub-function needs to be increased which in the result will increase the complexity of the hardware. In this thesis work, Parabolic Synthesis is combined with non-linear interpolation. In this case, only two sub-functions are required to get the same accuracy as in parabolic synthesis. So the equation (2.1) can be reduced to equation (2.32).

$$f_{org}(x) = s_1(x) \,.\, s_2(x) \tag{2.32}$$

This will decrease the hardware significantly. Another benefit of combining the parabolic synthesis with non-linear interpolation is that this approach will make it easy to adjust the error behavior of the approximation [7]. Therefore the first sub-function, $s_1(x)$, is used to calculate the initial value

of approximation and second sub-function, $s_2(x)$, is used to get the desired accuracy depending on the number of intervals used in the interpolation.

The approximation of the function can be implemented with two stages. The first stage is implemented according to first sub-function as shown in (2.5). The second stage can be implemented using non-linear interpolation as shown in (2.24). The first sub-function, $s_1(x)$ is constructed as parabolic synthesis as described in section 2.1.2 and second sub-function, $s_2(x)$ will be constructed as non-linear interpolation as described in section 2.2.2. The original function (2.32) will become (2.33).

$$f_{org}(x) = \left(x + (c_1 \,.\, (x - x^2)\right) \times \left(l_{2,i} + k_{2,i} \times x_w + c_{2,i} \times (x_w - x_w^2)\right) \tag{2.33}$$

In (2.33), *2,i* index represents the interval in which the interpolation is performed. The interval index, *i*, is a power of 2, which results in the number of intervals equal to 1, 2, 4, 8, and so on. The index *w* shows that the *x* term in the interpolation stage is dependent on the number of intervals. The *x* term is modified in such a way that when four intervals are used in the interpolation, then the two most significant bits are thrown away in the *x* term, i.e. 2 left shifts in the hardware. The truncation in (2.34) is performed in order to normalize the interval for second sub-function.

$$x_w = frac(2^w \,.\, x) \tag{2.34}$$

The removed integer part is used to decode in which interval of second sub-function the interpolation is performed. This integer part is used as an address to fetch the corresponding coefficients in the specific interval in the hardware.

The second sub-function is divided in partial sub-functions as shown in (2.35).

$$s_2(x) = \left\{ \begin{array}{ll} s_{2,0}(x_w), & 0 \le x < \frac{1}{2^w} \\ s_{2,1}(x_w), & \frac{1}{2^w} \le x < \frac{2}{2^w} \\ \quad\quad ... & \\ s_{2,I-1}(x_w), & \frac{I-1}{2^w} \le x < 1 \end{array} \right\} \tag{2.35}$$

As it can be seen that x is changed to x_w, which means that the partial sub-functions$s_{2,\mathrm{i}}(x_w)$ of second sub-function, $s_2(x)$, have equal range.

CHAPTER 3

3 Hardware Architecture

The hardware architecture of the methodology can be divided into three parts i.e., preprocessing, processing, and post processing. It was introduced by P.T.P Tang [1]. The preprocessing and post processing is the transformation stages and in processing part, the original function, $f_{org}(x)$, is calculated [16].

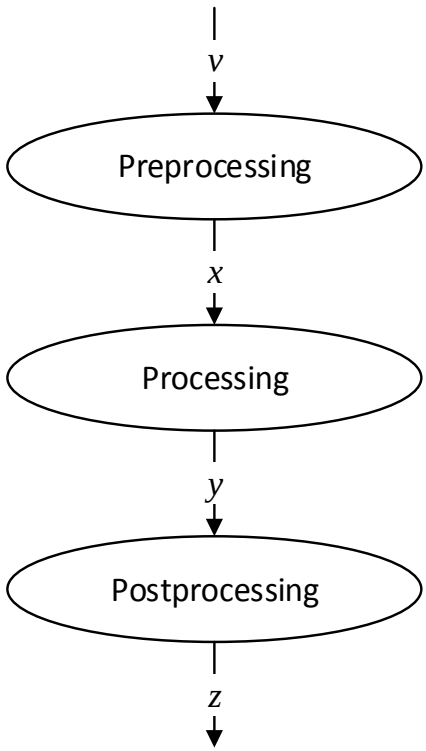


Figure 3.1:Three stage Architecture

3.1 Preprocessing

In the preprocessing part, the input signal *v* is normalized to prepare it for the processing part. For example an input signal sin(v) that lies between the interval 0 to $\frac{\pi}{2}$, will be normalized and converted into an output *x* that lies between the interval 0 to 1. This is performed by multiplying it with $\frac{2}{\pi}$ [16].

3.2 Processing

In the processing part, the original function, $f_{org}(x)$, is approximated that results in an output *y*. In this section the processing part for parabolic synthesis will be discussed first and then parabolic synthesis with non-linear interpolation will be discussed.

3.2.1 Parabolic Synthesis

Fig. 3.2 shows the basic architecture of the loop unrolled parabolic synthesis with four sub-functions. This architecture has an advantage of fast computation speed at the cost of large chip area [7].

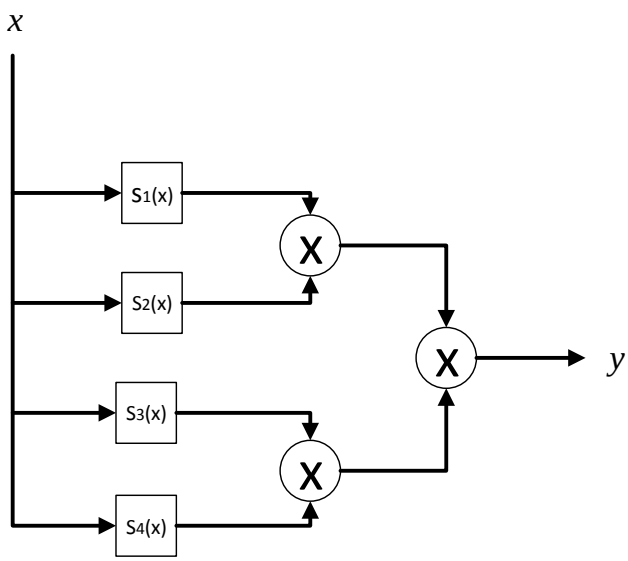


Figure 3.2: Basic hardware for loop unrolled architecture

The detailed hardware architecture of loop unrolled parabolic synthesis with four sub-functions is given in Fig. 3.3.

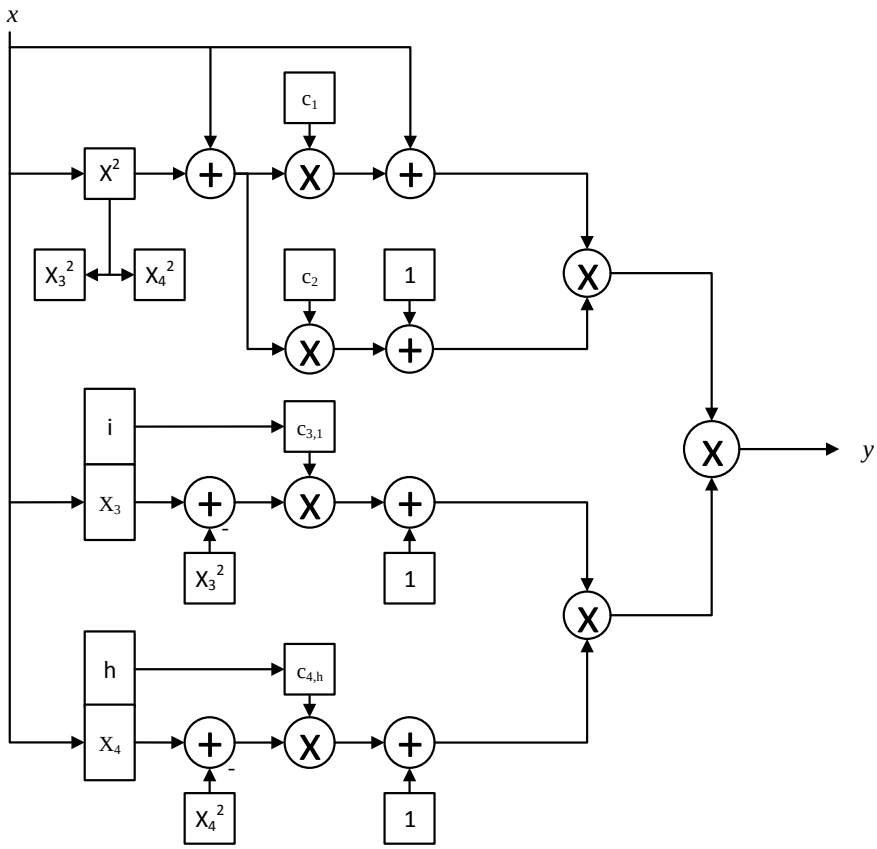


Figure 3.3: Detailed hardware architecture for 4 sub-function parabolic synthesis

In this architecture, (x-x^2) part is same for both first sub-function and second sub-function. The output of this part is multiplied with c_1 for first sub-function, $s_1(x)$, and with c_2 for second sub-function, $s_2(x)$ [7]. In the first sub-function, $s_1(x)$, after the multiplication with c_1, the x-value is added to it. However, in the second sub-function, $s_1(x)$, after the multiplication with c_2, a 1 is added. A special squaring unit is designed to calculate the partial products of x_3^2 and x_4^2. The latency and chip area can

be significantly reduced by designing this squaring unit, in comparison to using separate multipliers for each product. The index *i*, in the Fig. 3.3, are the most significant bits which help to determine the $c_{3.i}$ coefficient for the interval. Similarly, the index *h* in the fourth sub-function is the two most significant bits of *x* and it helps as an address for value of $c_{4.i}$ coefficients in the four intervals. The value of first and second sub-function is multiplied in parallel with the third and fourth sub-functions. The result of these two multiplications is multiplied with each other to compute the value of *y* [7].

3.2.2 Parabolic Synthesis with Non-Linear Interpolation

The processing part of parabolic synthesis combined with non-linear interpolation can be graphically visualized in Fig. 3.4. This architecture is designed to calculate a single function.

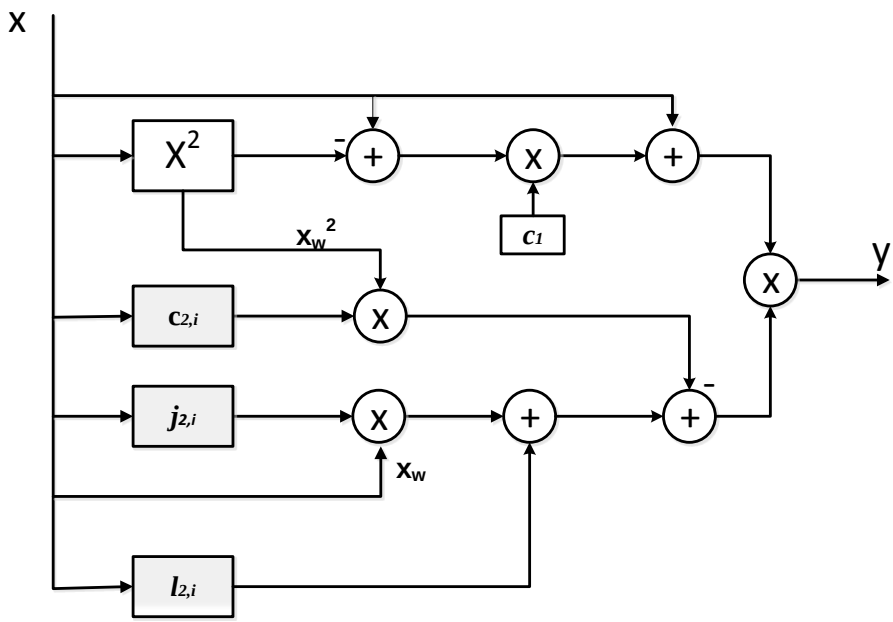


Figure 3.4: Architecture of parabolic synthesis with non-linear interpolation

The result of $(x - x^2)$ is multiplied with c_1 in the first sub-function, $s_1(x)$, and the result is added to x. As mentioned before, second sub-function is implemented as non-linear interpolation and it consists of three look-up tables, i.e. $l_{2,i}$, $j_{2,i}$ and $c_{2,i}$ for each interval i. The coefficient $j_{2,i}$ is multiplied with x_w which is the normalized value for corresponding interval. The results of this multiplication is added to $l_{2,i}$. The partial product of x^2, i.e. x_w^2 is multiplied with $c_{2,i}$. The result of this multiplication is subtracted from the result of addition of $(l_{2,i} + j_{2,i} \times x_w)$ [14]. The results of both sub-functions are multiplied with each other to compute the value of *y*.

The design contains four adders, four multipliers and one squarer block. Instead of using a multiplier a squarer block is specially designed to produce all the partial products needed to compute x^2 and x_w^2 [14]. A simplified version of a 6-bit squarer block can be seen in Fig. 3.5.

					x5	*x4*	*x3*	*x2*	*x1*	*x0*
					x5	*x4*	*x3*	*x2*	*x1*	*x0*
x5x4	*x5x3*	*x5x2*	*x5x1*	*x5x0*	*x4x0*	*x3x0*	*x2x0*	*x1x0*	*0*	*x0*
x5		*x4x3*	*x4x2*	*x4x1*	*x3x1*	*x2x1*		*x1*		
		x4	*x3x2*	*x3x2*		*x2*				

Figure 3.5: Specially designed 6-bit squarer

3.3 Post processing

The post processing stage is used to transform the value *z* from the output of processing stage i.e., *y* to the desired format in order to fulfill the approximation.

CHAPTER 4

4 Error Evaluation

The performance of any algorithm is characterized by its error behavior. Since the parabolic synthesis is an approximation based method, the error behavior holds a vital importance. An example of the error behavior for sine function using Parabolic Synthesis methodology is shown in Fig. 4.1.

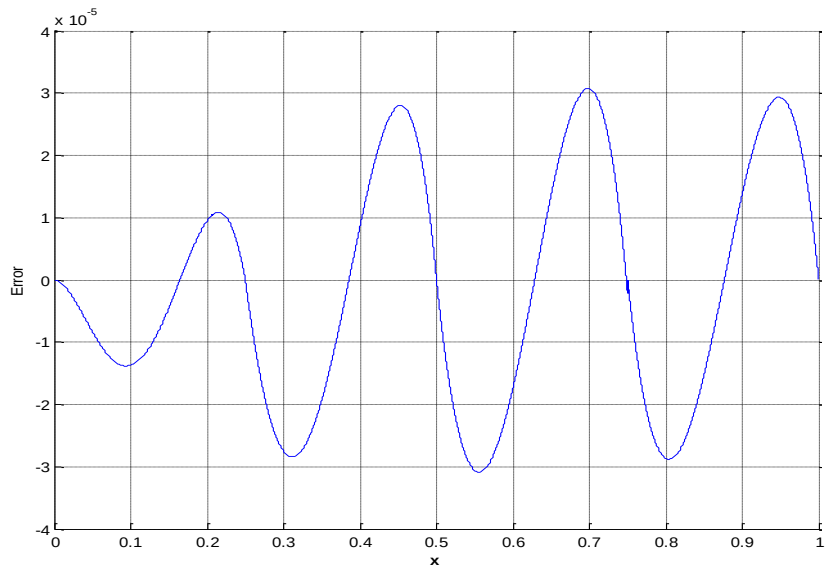


Figure 4.1: Error behavior for Parabolic Synthesis

There are five different metrics that can be used to characterize the error behavior [13] [14]. These metrics are as follow.

- Maximum Absolute Error

- Mean Error
- Standard Deviation
- Root-Mean-Square
- Median Error

4.1 Error Metrics

A brief description of the error metrics are given below. For detailed study, readers are referred to [12][13].

4.1.1 Maximum Absolute Error

The difference between the approximated value $\hat{x}_i$ and the actual value x_i is called the absolute error Δe_{abs}. Absolute error is shown in (4.1).

$$\Delta e_{abs} = |\hat{x}_i - x_i| \tag{4.1}$$

It is the maximum value that is calculated in the interval where the error is investigated [14].

4.1.2 Mean Error

For n numbers of separate values in a specific sequence of errors, the mean error $\bar{e}$ can be seen in (4.2).

$$\bar{e} = \frac{1}{n}\sum_{i=0}^{n}(\hat{x}_i - x_i) \tag{4.2}$$

In other words, it is the average of the absolute error of a sequence of n numbers [14].

4.1.3 Standard Deviation

The standard deviation is used to calculate the amount of change in a value from its expected value. The difference between standard deviation and average deviation is that the average value is calculated with power instead

of amplitude. In order to calculate the standard deviation, the deviations are squared before averaging. It is defined in (4.3) [14].

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=0}^{n}[\hat{x}_i - \bar{e}]^2} \tag{4.3}$$

4.1.4 Median Error

The median error $\tilde{x}$ is used to calculate the middle value for a given sequence of errors. If the sequence contains odd number of samples the median error $\tilde{x}$ is the middle sample and if the sequence contains even number or samples, median error $\tilde{x}$ is the mean of the two middle samples. For example, for a sequence $\{x_1, x_2, \dots, x_n\}$, the median error can be calculate as (4.4) and (4.5) [14].

$$\tilde{x} = x_{(\frac{n}{2}+1)} \text{ If } n \text{ is odd} \tag{4.4}$$

$$\tilde{x} = \left(x_{(\frac{n}{2})} + x_{(\frac{n}{2}+1)}\right) / \text{ If } n \text{ is even} \tag{4.5}$$

4.1.5 Root-Mean-Square

In order to calculate the deviation of a sinusoidal signal, Root-Mean-Square (RMS) value is used. This error metric is widely employed in electronics where both AC and DC values of a signal need to be measured. It is the square root of the average of squared difference between the approximated value $\hat{\boldsymbol{x}}_{\boldsymbol{i}}$ and the actual value x_i [14].

$$x_{rms} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{x}_i - x_i)^2} \tag{4.6}$$

4.2 Error Distribution

There are two development strategies that can be employed while developing an approximation. These are least square approximation and least maximum approximation. Least squares approximation is used to

minimize the average error and least maximum approximation is used in order to minimize the maximum error. Least square approximations are suitable when the approximated function is to be used in a series of computations. It is also important to investigate the error distribution so that the error of approximation is not of unilateral polarity [13] [14].

In order to evaluate error distribution evenness, standard deviation is compared with RMS. The error distribution is even if both the values are equal. The error behavior of sine function in Fig. 4.2 provides a good example of the error behavior methodologies explained in this Chapter. The manner of error distribution shows that the approximated value oscillates around the original function and is evenly distributed around zero [12] [13]. A diagram to visualize the error distribution is shown in Fig. 4.2.

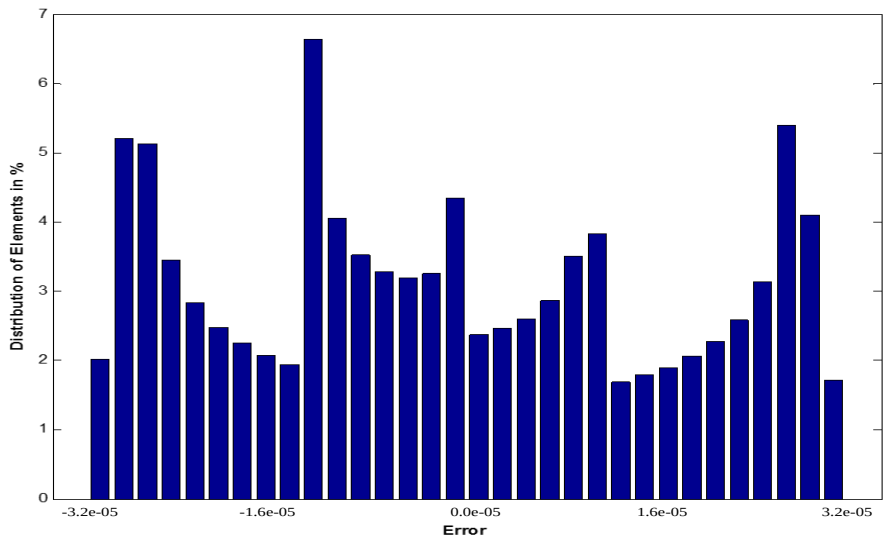


Figure 4.2: The distribution of error between original function and the approximation

CHAPTER 5

5 Architecture and Coefficients Approximation

The objective of this thesis work is to design and implement the approximation of the sine and cosine functions in all quadrants i.e., 360˚. The approximation is implemented using two stages of parabolic synthesis. The first stage is implemented using parabolic synthesis methodology and second stage is implemented as a non-linear interpolation as described in section 3.2. In this chapter, the hardware architecture to implement sine and cosine functions using parabolic synthesis and non-linear interpolation technique will be discussed. A methodology is also described to calculate the coefficients for the second stage of approximation, i.e. non-linear interpolation.

5.1 Architecture

As described in chapter 3, the hardware architecture of the methodology consists of three parts i.e., preprocessing, processing, and post processing. This architecture will compute the sine and cosine functions based on the input signal v and produce the output z_{sine} *and* z_{cosine}. The block diagram of the architecture is given in Fig. 5.1.

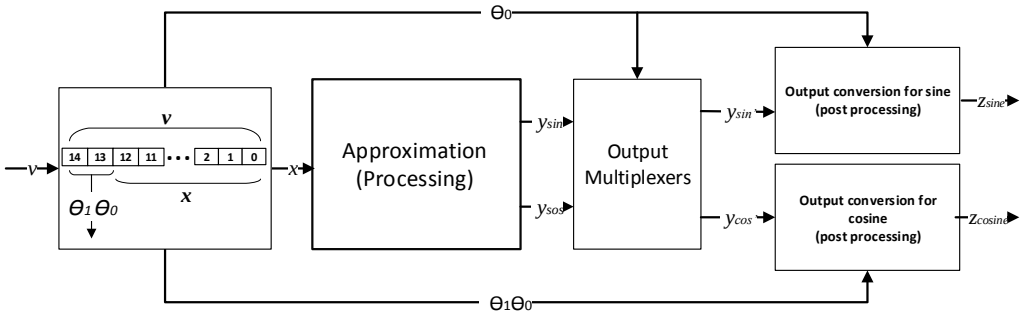


Figure 5.1: Block diagram of the architecture

As shown in the Fig. 5.1 the normalized input signal *v* in the interval, 0 to 2π, converted into the input *x*. The two most significant bits, $\theta_1\theta_0$, of the input signal, *v*, are taken away and used as an enable signal for the output multipliers and two's conversions. The rest of the bits are used as input signal, *x*, for the approximation (processing) block. The processing block performs the approximation and multiplications for the sub-functions of sine and cosine approximations. The approximated output, y_{sin} and y_{cos}, from the processing block goes to the output multiplexers and new, y_{sin}' and y_{cos}', are chosen depending on the input quadrant. The sign of the new, y_{sin}' and y_{cos}', values are changed in the output conversion blocks by using, $\theta_1\theta_0$, as enable signals to produce the output , z_{sin} and z_{cos}.

5.1.1 Preprocessing

A normalized input to the system, *v*, is expressed in 15 bits, which means that the input signal is divided in 0 to $2^{15} - 1$ steps. The maximum input to the system is '111111111111111_2' which corresponds to a normalized angle of 3.99999 in decimal. Therefore, the function of pre-processing block is to remove the two MSBs (integer part) and send the rest of the bits as *x* value to the processing part.

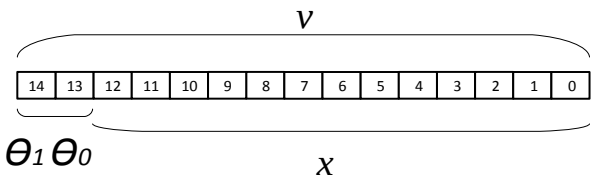


Figure 5.2: Pre processing block

5.1.2 Processing

In the processing part, the original function, $f_{org}(x)$, is approximated that results in output y_{sin} and y_{cosine}. In this architecture, only two sub-functions are required to get the same accuracy as in parabolic synthesis. Therefore the equation (2.1) can be reduced to (5.1).

$$f_{org}(x) = s_1(x) \,.\, s_2(x) \tag{5.1}$$

The approximation of sine and cosine functions is given in (5.2) and (5.3). The angle x is the normalized fractional part of v. It can be seen that only the first sub-function, $s_1(x)$, differs for both sine and cosine functions [14].

$$s_{1,sin}(x) = (x) + c_1 \times (x - x^2)$$

$$s_{2,sin}(x) = l_{2,is} + k_{2,is} \times x_w + c_{2,is} \times (x_w - {x_w}^2) \tag{5.2}$$

$$f_{org}(x)_{sin} = s_{1,sin}(x) \times s_{2,sin}(x)$$

$$s_{1,cos}(x) = (1 - x) + c_1 \times (x - x^2)$$

$$s_{2,cos}(x) = l_{2,ic} + k_{2,ic} \times x_w + c_{2,ic} \times (x_w - {x_w}^2) \tag{5.3}$$

$$f_{org}(x)_{cos} = s_{1,cos}(x) \times s_{2,cos}(x)$$

The original function, $f_{org}(x)$, for both sine and cosine will become as shown in (5.4) and (5.5) respectively.

$$f_{org}(x)_{sin} = \left(x + (c_1 . (x - x^2)\right) \times \left(l_{2,is} + k_{2,is} \times x_w + c_{2,is} \times xw - xw2 \quad (5.4)$$

$$f_{org}(x)_{cos} = \left(1 - x + (c_1 . (x - x^2)\right) \times \left(l_{2,ic} + k_{2,ic} \times x_w + c_{2,ic} \times xw - xw2 \quad (5.5)$$

It can be seen that both the first and second sub-functions for sine and cosine are identical. There is one extra subtraction in the first sub-function for cosine. The second sub-functions for sine and cosine are similar and the only difference in second sub-functions is that they use different set of coefficients. Therefore both the sub-functions can be combined in parallel. The multiplications of these sub-functions with their corresponding sub-functions produce the result for sine and cosine functions simultaneously. In this way, the hardware for a multiplier, adder and another special squarer can be saved.

5.1.3 Post Processing

In the post processing block, the output from the processing block is converted in order to get the desired results. The output of the processing block, y_{sin} and y_{cos}, are the approximated result from the processing stage in the range 0 to 1 for an input *x*. However, the actual quadrant of any output is unknown since the computations are performed in first quadrant. The output, y_{sin} and y_{cos}, has to be transformed back to their actual values in their respective quadrants which is determined using $\theta_1\theta_0$ bits that come from preprocessing block.

In order to change the output from processing block to its corresponding quadrant, for both sine and cosine, output multiplexers are used that determine the new, y_{sin}' and y_{cos}', values based on the input quadrant. The

input quadrant is determined using, θ_0, as enable signal for multiplexers. The sign of the new, y_{sin}' and y_{cos}', values needs to be changed as well. The sine function is positive in first and second quadrant, therefore, no conversion is needed. However, it is negative is third and fourth quadrant, therefore the sign needs to be changed. This is achieved by a two's complement conversion at the final stage, where $\theta_1$is used as an enable signal for two's complement conversion.

Similarly, cosine function is positive in first and the fourth quadrant and negative in second and third quadrant, therefore, we need to change the sign of the y_{cos}' value for the second and third quadrants. This conversion can easily be performed by using $\theta_1 \text{ XOR } \theta_0$ as a control signal for the sign conversion in the respective quadrants. The Table I shows when we need to transform the outputs for sine and cosine depending on the integer part, $\theta_1\theta_0$, coming from the preprocessing stage [14].

TABLE I: OUTPUT TRANSFORMS

	Quadrant 1	**Quadrant 2**	**Quadrant 3**	**Quadrant 4**
Sine	+	+	-	-
Cosine	+	-	-	+

The architecture of two's complement conversion for sine function is shown in Fig. 5.3. Half adders (HAs) and XOR gates are used in the architecture. For example, in order to calculate the *z* value for trigonometric identities, a control signal θ_1 or $\theta_1 \text{ XOR } \theta_0$ will be used for the conversion of sine or cosine function respectively [14].

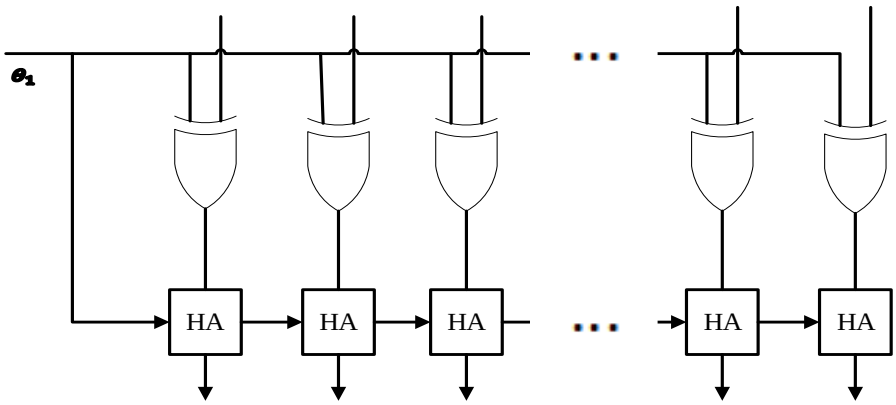


Figure 5.3: Two's complement architecture for sine function

5.2 Coefficients Approximation

In order to implement the approximation of trigonometric functions, sine and cosine, we need to develop the first help function. The first help function, $f_1(x)$, is the function from which the non-linear interpolation is developed from. The first help function, for the sine function, is developed according to (5.6).

$$f_1(x) = \frac{\sin\left(\frac{\pi}{2} \times x\right)}{x + \left(\frac{\pi}{2} - 1\right) \times (x - x^2)} \tag{5.6}$$

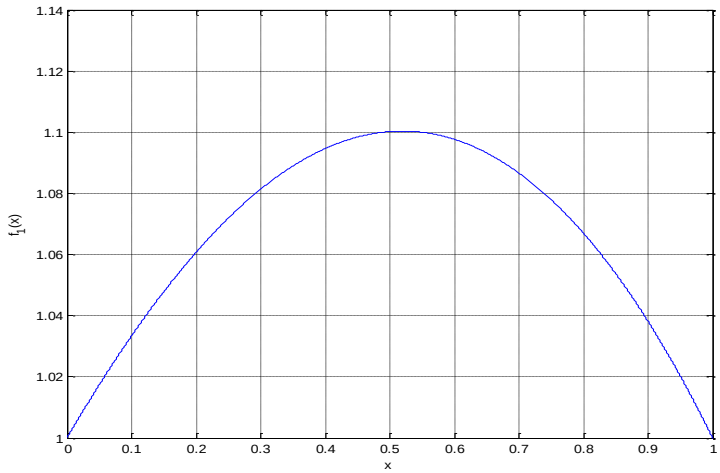


Figure 5.4: First help function, $f_1(x)$.

5.2.1 Linear part

The linear part of the interpolation consists of two coefficients for each interval, a starting point,$l_{2,i}$, and a gradient, $k_{2,i}$. In (2.24), $l_{2,i}$ is the starting point of an interval of the interpolation, which is computed by inserting the value of x for the starting point of the interval, $x_{start,i}$, in the first help function $f_1(x)$[14].

$$l_{2,i} = f_1(x_{start,i}) \tag{5.7}$$

In (2.24), $k_{2,i}$ is the gradient for an interpolation interval. The gradient k_i for an interval is computed as the end point value of the function $f_1(x_{end,i})$, subtracted with the start point value of the function $f_1(x_{start,i})$ of an interval. Since the interval is normalized to one, no denominator is needed, as shown in (5.8) [14].

$$k_i = f_1(x_{end,i}) - f_1(x_{start,i}) \tag{5.8}$$

The coefficients for the linear part of the interpolation are calculated according to (5.7) and (5.8) for four intervals, i.e. $i = 2$. The result of linear interpolation is shown in Fig. 5.5.

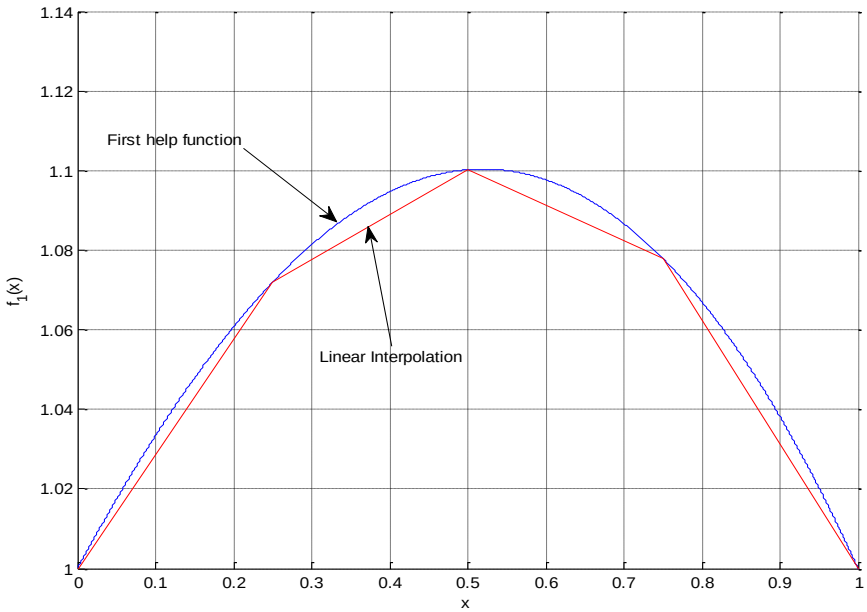


Figure 5.5: First help function and the linear interpolation of the first help function.

5.2.2 Non-linear part

In (2.24), $c_{2,i}$ is pre-computed so that the sub-function for the interval i, $s_2(x_w)$, cuts the function $f_1(x)$, in the middle of the interval i when $x_w = 0.5$, which satisfies the point $x_{middle,i}$ for $f_1(x)$, as shown in (5.9).

$$c_{2,i} = 4 \times \left(f_1\left(x_{middle,i}\right) - l_{2,i} - k_{2,i} \times 0.5\right) \tag{5.9}$$

If we subtract the linear interpolation of first help function from the first help function, it will generate a function with a parabolic looking function

in each interval as shown in Fig. 5.6. The coefficients for the non-linear part of the interpolation are calculated in according to (5.9).

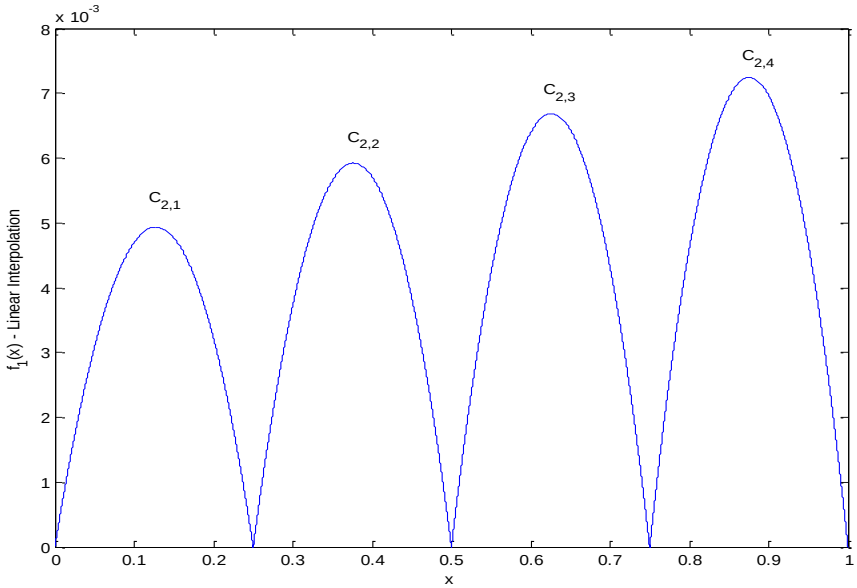


Figure 5.6: Approximation of the difference of first help function subtracted with the linear interpolation of first sub-function

The peak value of each curve represents the corresponding $c_{2,i}$ coefficients of each interval. The rest of the coefficients, i.e. $l_{2,i}$, $j_{2,i}$, and $k_{2,i}$ are also calculated using the equations (5.7), (5.8), and (5.9) . Similarly, the coefficients for cosine function can also be calculated in Matlab [14]. The approximated coefficient values for both sine and cosine function are listed in Table II and Table III respectively.

TABLE II: COEFFICIENT VALUES FOR SINE FUNCTION

Coefficients	Value in decimal	
c_1	$c_1 = 0.570796326794897$	
$c_{2,i}$	$c_{2,1} = 0.0199535148492645$,	$c_{2,2} = 0.0237052129326976$,
	$c_{2,3} = 0.0267498465570153$,	$c_{2,4} = 0.0289774221630130$
$l_{2,i}$	$l_{2,1} = 1.000000000000000$,	$l_{2,2} = 1.071895094550420$,
	$l_{2,3} = 1.100234394764010$,	$l_{2,4} = 1.078018196966255$
$k_{2,i}$	$k_{2,1} = 0.287477578201686$,	$k_{2,2} = 0.113380000854360$,
	$k_{2,3} =$-0.088843391191025,	$k_{2,4} =$ -0.312024187865020

TABLE III: COEFFICIENT VALUES FOR COSINE FUNCTION

Coefficients	Value in decimal	
c_1	$c_1 = 0.570796326794897$	
$c_{2,i}$	$c_{2,1} = 0.0289774221630130$,	$c_{2,2} = 0.0267498465570153$,
	$c_{2,3} =$0.0237052129326976,	$c_{2,4} = 0.0199535148492645$
$l_{2,i}$	$l_{2,1} = 1.000000000000000$,	$l_{2,2} = 1.078018196966255$,
	$l_{2,3} = 1.100234394764010$,	$l_{2,4} = 1.071895094550420$
$k_{2,i}$	$k_{2,1} = 0.312024187865020$,	$k_{2,2} = 0.088843391191025$,
	$k_{2,3} =$-0.113380000854360,	$k_{2,4} =$ -0.287477578201686

These coefficients are used to approximate the sine and cosine functions. The design and hardware implementation of these functions using the coefficients values given in Table II and Table III is explained in chapter 6.

CHAPTER 6

6 Hardware Design

In this thesis work, Parabolic Synthesis is combined with non-linear interpolation to implement the approximation of sine and cosine functions. The design is implemented using two stages of parabolic synthesis, i.e., parabolic synthesis and non-linear interpolation as discussed in chapter 5. In this chapter, the hardware structure of the combined methodology is discussed. There are two sub-functions that are used to get the same accuracy as in parabolic synthesis. Therefore the equation for original function, $f_{org}(x)$, can be written as

$$f_{org}(x) = s_1(x) \, . \, s_2(x) \tag{6.1}$$

The first sub-function, $s_1(x)$, is constructed as parabolic synthesis as described in section 2.1.2 and second sub-function, $s_2(x)$, will be constructed as non-linear interpolation as described in section 2.3.

The hardware design is divided into three different parts, i.e. preprocessing, processing and post processing. In the preprocessing part the two most significant bits are removed from the signal. The implementation of approximation of the original function, $f_{org}(x)$, is performed in processing part. In this part, the parabolic synthesis is combined with non-linear interpolation [14]. In the post processing part, the output from the processing block is converted back to its original value.

6.1 Preprocessing

The parabolic synthesis uses the already normalized input *v*. As described in section 5.1.1, the input transformation is performed in the pre processing block where the integer part i.e., two most significant bits, $\theta_1\theta_0$, are taken away. θ_0 is used as an input for the multiplexer to select the corresponding output for the multiplexer depending on the input quadrant. This integer part, $\theta_1\theta_0$, is also used as an enable signal to determine the two's complement transformation in the post processing stage.

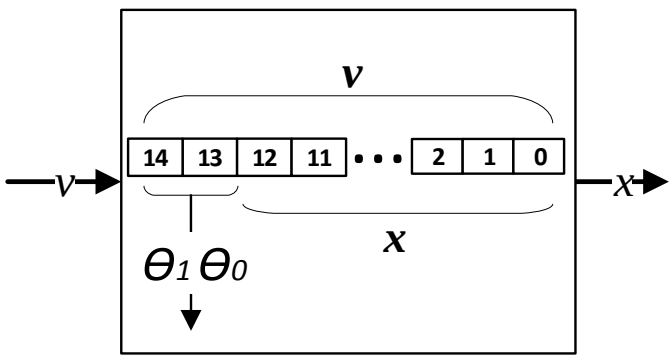


Figure 6.1: Pre processing block

6.2 Processing

When performing the approximation of the sine and cosine functions only the approximation of first quadrant needs to be done. In order to design the whole unit circle, the first quadrant of function can be reused with some additional hardware. The first help function, $s_1(x)$, for sine and cosine is shown in equation (6.2) and (6.3) [14].

$$s_1(x)_{sin} = x + (c_1 \,.(x - \, x^2)) \tag{6.2}$$

$$s_1(x)_{cos} = (1 - x) + (c_1 \,.(x - \, x^2)) \tag{6.3}$$

It can be seen that both the sub-functions can be combined in parallel to produce the result for sine and cosine functions simultaneously. It should be noted that the c_1 coefficient for both sine and cosine is same. The architecture for calculating the first sub-function for sine and cosine functions based on parabolic synthesis methodology is shown in Fig. 6.2.

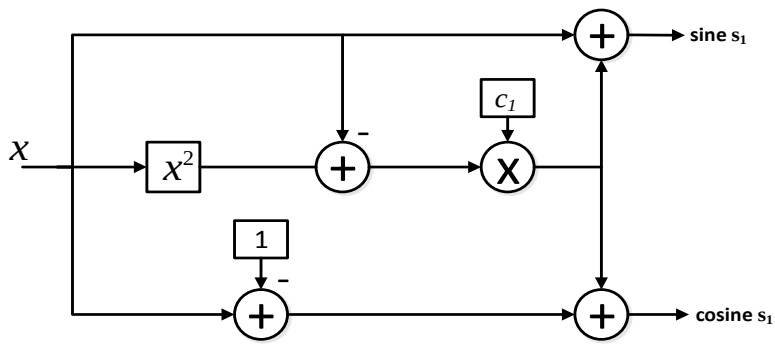


Figure 6.2: First sub-function architecture for sine and cosine

The calculation of the first coefficient c_1 for the since function is shown in (6.4) [14].

$$c_1 = \lim_{x \to 0} \frac{sin\left(\frac{\pi}{2} . x\right)}{x} - 1 = \frac{\pi}{2} - 1 \tag{6.4}$$

The c_1 multiplication in Fig. 6.2, uses a fixed multiplier so it can be replaced with simple shift and add operations. In the same way, the addition of "1" is simply a matter of routing wire in hardware [14].

Since we have power of two numbers, we can use the left fractional bits as address bits to look-up-table for the coefficient selection. The bits can be separated by x AND no-of-bits. However, in hardware it is simply a question of routing wires. For example, if we have two intervals, we separate one bit only, i.e. if the fractional MSB bit is "0", the left interval is addressed and if the MSB is "1", the right interval is used. For four

intervals, we get the four addresses $t =$ "00", "01". "10", and "11", e.g. if we have $x = 1010001_2$, the third interval will be addressed [14].

The remaining bits are used as a new "*x*-value", which is $x_w = (x - t)$, where *t* is the two MSBs of *x*. For the above example, $x = 1010001_2$, we thus get $x_w = 10001_2$, which are the remaining bits of x shifted two times to the right. The *t* bits will be used as address bits for the coefficient i.e., $l_{2,i}$, $j_{2,i}$, and $c_{2,i}$ tables.

The second help function $s_2(x)$ for both sine and cosine will remain the same and is shown in (6.4) and (6.5).

$$s_{2,sin}(x) = l_{2,is} + k_{2,is} \times x_w + c_{2,is} \times (x_w - {x_w}^2) \tag{6.4}$$

$$s_{2,cos}(x) = l_{2,ic} + k_{2,ic} \times x_w + c_{2,ic} \times (x_w - {x_w}^2) \tag{6.5}$$

The term x_w^2 is the square of partial product which comes from the special squarer designed in the project to produce the outputs x^2 and x_w^2 simultaneously. Similar to first sub-function, the hardware of second sub-function can also be joined to share some part of hardware. In this way, the area for an adder can be saved. Fig. 6.3 shows the second sub-function in the improved architecture, based on non-linear interpolation [14].

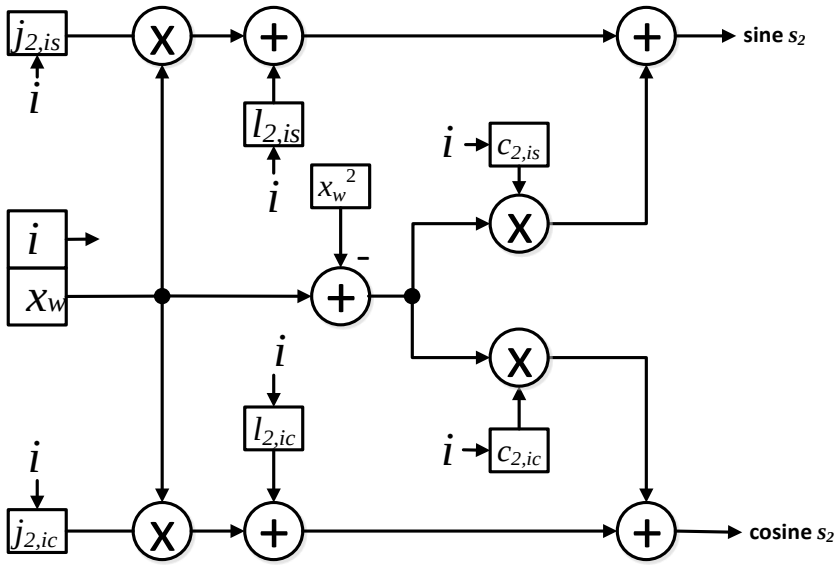


Figure 6.3: Hardware design for combined second sub-function

Finally, the outputs from first sub-function block and second sub-function block are multiplied together to calculate the output of processing block for both sine and cosine simultaneously.

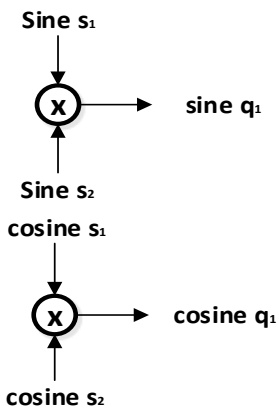


Figure 6.4: Multiplication of outputs from first and second sub-function blocks

6.3 Post Processing

As explained in section 5.1.3, all the calculations are performed in first quadrant. Therefore the output needs to be transformed back to their actual quadrants. This is achieved by transforming the outputs $sineq_1$ and $cosineq_1$ from the processing to their original quadrant. This is done by using a multiplexer and using θ_0 as an enable signal as shown in Fig. 6.5. the sign of the output from these multiplexers is changed by performing two's complement conversion. For the cosine output $\theta_1 XOR\, \theta_0$ is used as an enable signal to ensure that the cosine output is positive in first and fourth quadrant and negative in second and third quadrant. Similarly, θ_1 is used as an enable signal for the two's complement conversion for the sine signal which ensures that the sine is positive for sign of the output is positive in first and second quadrant and negative in third and fourth quadrant.

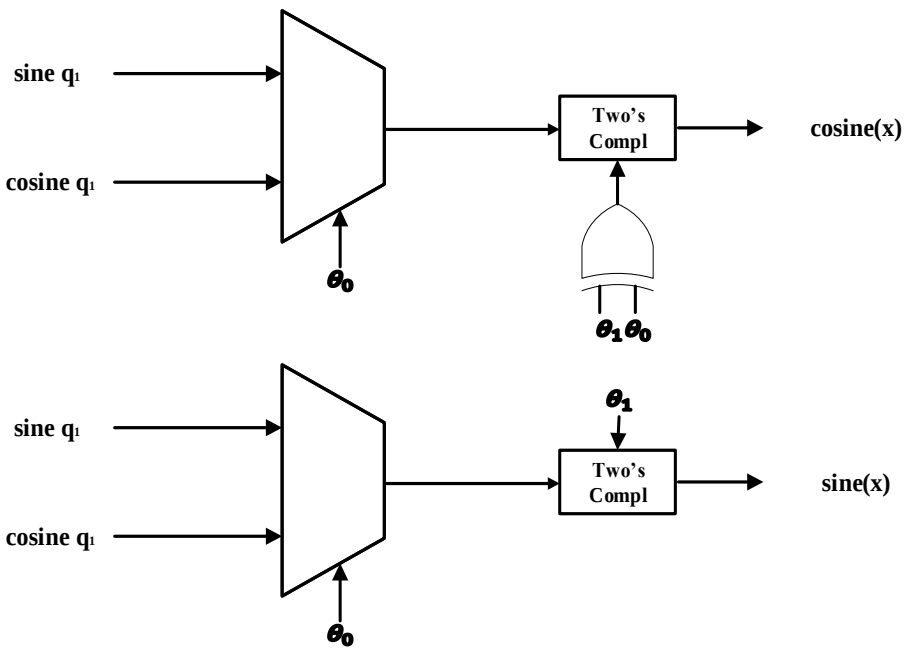


Figure 6.5: Post processing architecture for all four quadrants

6.4 Final Architecture

In order to compute the sine and cosine approximations, the architecture in Fig. 6.6 is used in the thesis work [14].

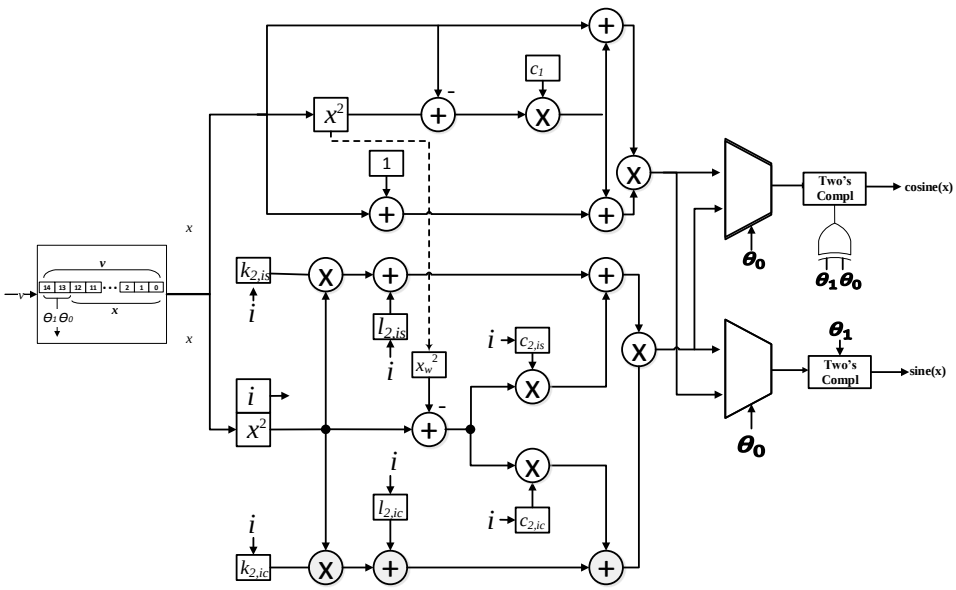


Figure 6.6: The final architecture

The architecture consist of multipliers, one special squarer block, adders, two two's conversion converters, and two multiplexers. The input, *x*, from preprocessing block goes to first and second sub-function blocks i.e., the processing part. The output for first sub-function for both sine and cosine functions is multiplied with the respective output from the second sub-function block. These multiplications produce intermediate results, sineq_1 and cosineq_1 from processing block. These intermediate values need to be converted into the desired results, which depends on the transformation of the quadrant in preprocessing stage. Therefore, two multiplexers are used the convert them into their respective quadrants and two's complement

conversion is performed in order to change their signs, in the post processing stage, to get the final results.

The critical path of the design is given in Fig. 6.7.

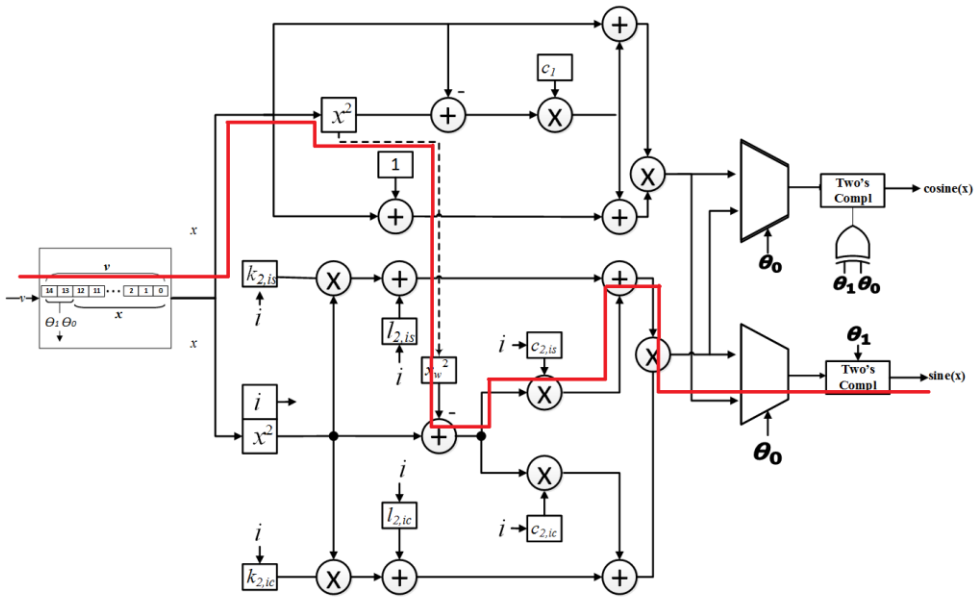


Figure 6.7: Critical path of the design

The critical path of the hardware goes through

- One squarer
- Two multipliers
- Two adders
- One Multiplexer
- One two's conversion converter

6.5 Word Lengths

The input word length for the hardware design is 15 bits. As shown in (6.6), all possible input values should be tested at the end.

$$\theta_f = 0 \text{ to } 2^{15} - 1 = 0 \text{ to } 32768 - 1 \tag{6.6}$$

For hardware design, these integer values are not longer than 15 bits and they are not needed to be truncated. However, the values needs to be scaled down to a 0 to 90 degree scaled as shown in (6.7) [14].

$$90 \times (\theta_f - 1)/2^{15} \tag{6.7}$$

Since 90 degrees are not allowed, the maximum input value is shown in (6.8).

$$\frac{2^{15}-1}{2^{15}} = 89.99725341796875_{10} = 111111111111111_2 \tag{6.8}$$

All the operations in VHDL are performed in floating point and the numbers are expressed as signed. Therefore it will add an extra bit to all the signals going to adders. Fig. 6.8 shows the internal word lengths of all the signals in the hardware design.

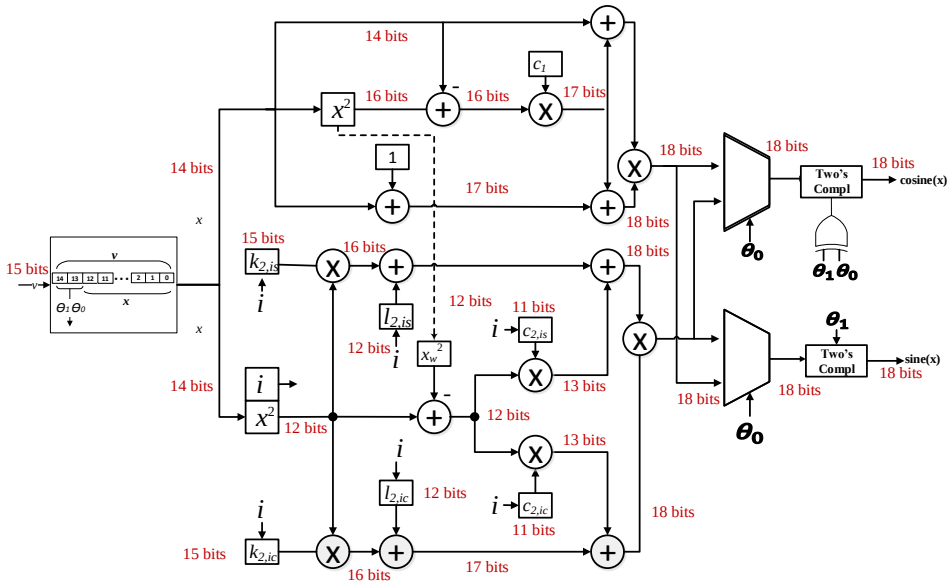


Figure 6.8: Internal word lengths of the design

The word lengths of the coefficients in Table II and Table III can also be optimized. The $l_{2,i}$ coefficients are greater than 1 so there will be 16 bits needed to express them in binary numbers plus an extra bit for signed number. However, when these numbers are truncated and converted into binary number there are many zeroes in the LSBs. These zeroes can be ignored in hardware which leaves 12 bits representation for $l_{2,i}$ coefficients. A 15-bit signed representation is used for $k_{2,i}$ coefficients and $C_{2,i}$ coefficients are expressed in 11-bit signed numbers. This will help greatly to reduce the area for multipliers and adders. The optimized and truncated coefficient values for sine and cosine functions are given in Table IV and Table V respectively.

TABLE IV: TRUNCATED COEFFICIENT VALUES FOR SINE FUNCTION

Coefficients	Value in decimal	
c_1	c_1 =0.570556640625	
$c_{2,i}$	$c_{2,1}$ = 0.01953125,	$c_{2,2}$ = 0.0234375,
	$c_{2,3}$ = 0.0263671875,	$c_{2,4}$ = 0.02880859375
$l_{2,i}$	$l_{2,1}$ = 1.0000000000,	$l_{2,2}$ = 1.07177734375,
	$l_{2,3}$ = 1.1009765625,	$l_{2,4}$ = 1.07763671875
$k_{2,i}$	$k_{2,1}$ = 0.287506103515625,	$k_{2,2}$ = 0.1134033203125,
	$k_{2,3}$ =-0.088836669921875,	$k_{2,4}$ = -0.312042236328125

TABLE V: TRUNCATED COEFFICIENT VALUES FOR COSINE FUNCTION

Coefficients	Value in decimal	
c_1	c_1 =0.570556640625	
$c_{2,i}$	$c_{2,1}$ = 0.02880859375,	$c_{2,2}$ = 0.0263671875,
	$c_{2,3}$ = 0.0234375,	$c_{2,4}$ = 0.01953125
$l_{2,i}$	$l_{2,1}$ = 1.00000000000000,	$l_{2,2}$ =1.0780029296875,
	$l_{2,3}$ = 1.10020446777344,	$l_{2,4}$ = 1.07186889648438
$k_{2,i}$	$k_{2,1}$ = 0.312042236328125,	$k_{2,2}$ = 0.088836669921875,
	$k_{2,3}$ =-0.1134033203125,	$k_{2,4}$ = -0.287506103515625

CHAPTER 7

7 Implementation and Error Behavior

Based on the methodology described in Chapter 2, 3, 5, and 6, a reference model for the approximation is implemented in MATLAB and implemented in hardware using VHDL. In this way the functional behavior is of the design is verified. The coefficients in Table IV and Table V are used in the design. Fig. 7.1 shows the approximated sine and cosine functions and their error behavior in decibel.

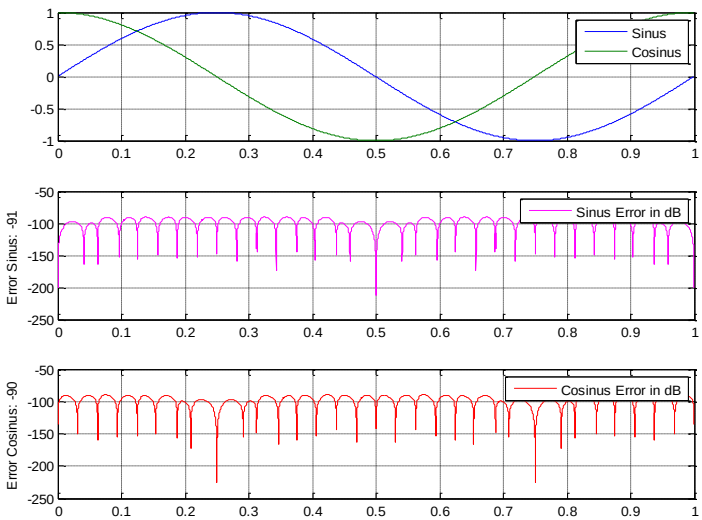


Figure 7.1: Approximation of sine and cosine functions and the error

7.1 Optimization

In order to increase the accuracy of the approximation, the coefficients, $l_{2,i}$, $j_{2,i}$, and $k_{2,i}$ in the second sub-function, $s_2(x)$, need to be optimized. The optimization helps to characterize the behavior of the error. The optimization must be performed in parallel with the truncation and the evaluation of word lengths. For a better understanding, truncation effects are not taken into consideration in this section.

The second sub-function is given in (7.1).

$$s_2(x) = l_{2,i} + k_{2,i} \times x_w + c_{2,i} \times (x_w - x_w^2) \tag{7.1}$$

The optimization strategy can be performed on all 12 coefficients of the second sub-function, $s_2(x)$, using four intervals. Since the coefficients $c_{2,0}$ through $c_{2,3}$ adjust the height of the parabolic part of the second sub-function, the optimization is primarily performed on these coefficients.

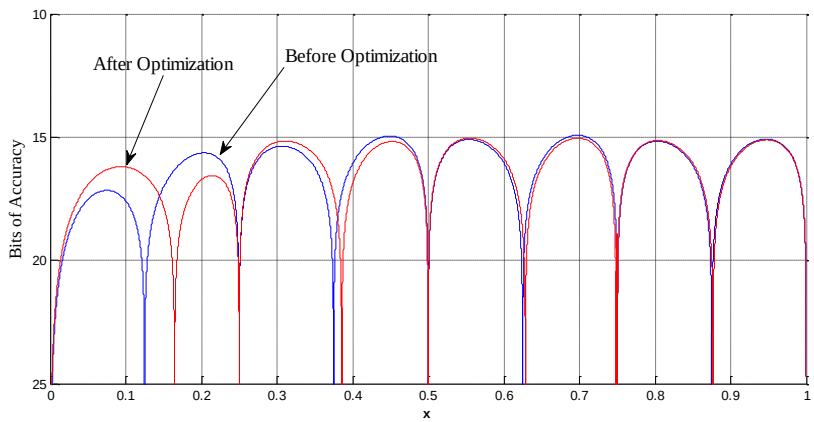


Figure 7.2: The absolute accuracy in bits of approximation, before and after optimization

As it can be seen in Fig. 7.2, there is a reduction of a half bit for the largest error in the interval, $0 \leq x < 0.25$. However, there is a negligible improvement in terms of largest error of approximation. During the hardware design, the optimization is performed on bit level [7].

7.2 Truncation

All the coefficients and signals in the MATLAB reference model need to be truncated since it is implemented exactly like the hardware architecture implemented in VHDL. The word length of the coefficients can be optimized in such a way that the system does not lose its precision. All the signal need to be truncated in such a way that the MATLAB model is an exact mirror of ASIC implementation. For example, a calculation

$$lin = c_2 \times (x_w - x_w^2)$$

Should be implemented like this

$$c_2 = \text{trunc}(c_2)$$

$$x_2 = \text{trunc}(x_w^2)$$

$$A = \text{trunc}(x - x_2)$$

$$lin = \text{trunc}(c_2 \times A)$$

7.3 Error Behavior

In order to provide the greater resolution and better understanding of the results, a logarithmic scale is used. The logarithmic unit is decibel (dB) and the binary numbers can be related to each other as shown in (7.2).

$$20\log(2) = 20 \times (0.301) \approx 6dB \tag{7.2}$$

This shows that 6dB is equal to 1 bit of resolution. For example, an error of 0.001 is same as 20log (0.001) = 20*(-3) = -60dB. We can transform it into bits, which gives the error 60/6 = 10 bits or less [14].

The error behavior of the Parabolic Synthesis combined with Non-Linear Interpolation can be seen in Fig. 7.3. The error is calculated by subtracting the sine function approximation from the original sine function after the truncation.

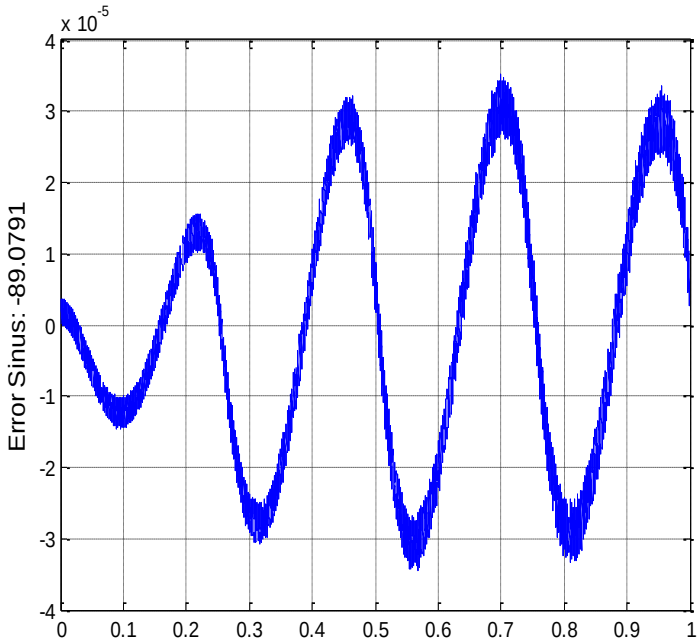


Figure 7.3: Error behavior of sine function after truncation

It should be noted that the approximated value oscillates around the original function in the desired manner and it confirms that the error is evenly distributed around zero.

TABLE VI: THE ERROR METRICS FOR THE TRUNCATED AND OPTIMIZED IMPLEMENTATION

Error Metrics	Value	Bits
Maximum Absolute Error	0.00003600399789538	14.84
Mean Error	-0.0000006352127232	20.65
Median	-0.0000018745591403	
Standard Deviation	0.00001891092717920	
Root Mean Square	0.00001891104738872	

Table VI shows that the resolution of the algorithm is almost 14.84 bits, which is very close to the required resolution for this thesis work. The mean error is very small. It should be noted that the standard deviation and root mean square values are almost identical which indicates that the error of approximation is evenly distributed around zero.

CHAPTER 8

8 Results

The approximation for Parabolic Synthesis and Non-Linear interpolation is implemented in stm65 CMOS technology. The design is simulated and compared for speed, area, and power consumption. The design is implemented in VHDL and the synthesized code is simulated for different standard libraries in Design Vision. Low Power High V_t (LPHVT) and Low Power Low V_t (LPLVT) transistors are used in with different supply voltages, V_{DD}= $\{1.00, 1.10, 1.20\}$ volts. This chapter describes the speed, area, and power consumption of the system and comparison with other methodologies.

8.1 Synthesis

The synthesis is performed in a design tool called Design Vision by Synopsis. During the synthesis a gate level netlist is generated from the VHDL design using STMicroelectronics 65nm Technology. This netlist is analyzed for speed, area, and power consumption. The results of different parameters are described below.

8.1.1 Area Results

The minimum area of the design is estimated by setting the area design constraint to zero in Design Vision. The minimum area results of the design for different libraries of Low Power High V_t (LPHVT) and Low Power Low V_t (LPLVT) for supply voltages, V_{DD}= $\{1.00, 1.10, 1.20\}$ volts are given in the Table VII.

TABLE VII: MINIMUM AREA RESULTS FOR LPHVT AND LPLVT

	LPHVT			LPLVT		
Voltage (V)	**1.00**	**1.10**	**1.20**	**1.00**	**1.10**	**1.12**
Area (μm^2)	15953	15974	15966	16132	16592	17056

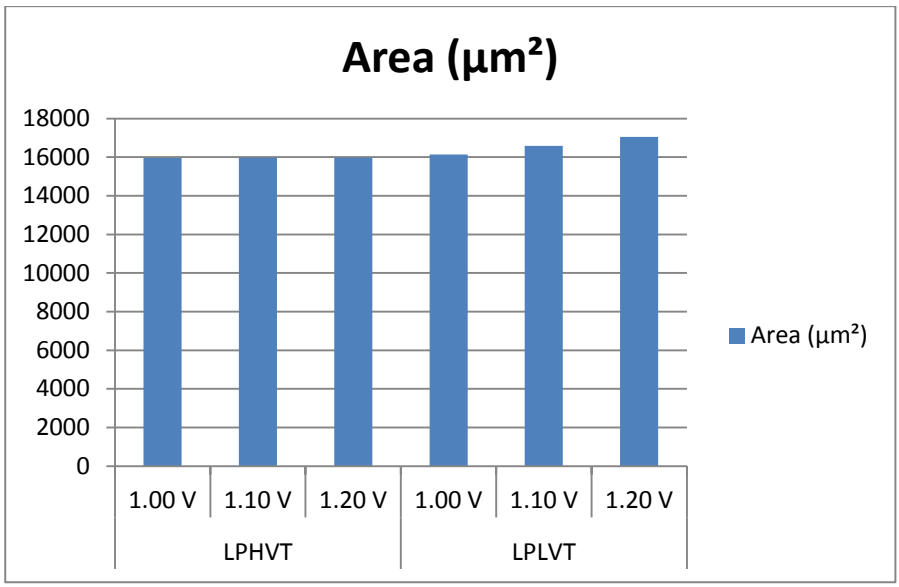


Figure 8.1: Minimum area results in a bar graph

The area for different sub-functions and output multiplier can be seen in the Table VIII: The Area Results for Individual Blocks in Design for LPHVT @ 1.2 Volts. The synthesis is performed with Low Power High V_t (LPHVT) library at a supply voltage of V_{DD}= 1.2 volts.

TABLE VIII: THE AREA RESULTS FOR INDIVIDUAL BLOCKS IN DESIGN FOR LPHVT @ 1.2 VOLTS

Block	Area (µm²)	Percentage (%)
First Sub-function	2433	15.23
Second Sub-function	8448	52.91
Output Multipliers	3992	25
Output Conversions	575	3.6
In/out Registers	518	3.24
Total	15966	100

For better understanding the individual blocks can be identified in Fig 8.2.

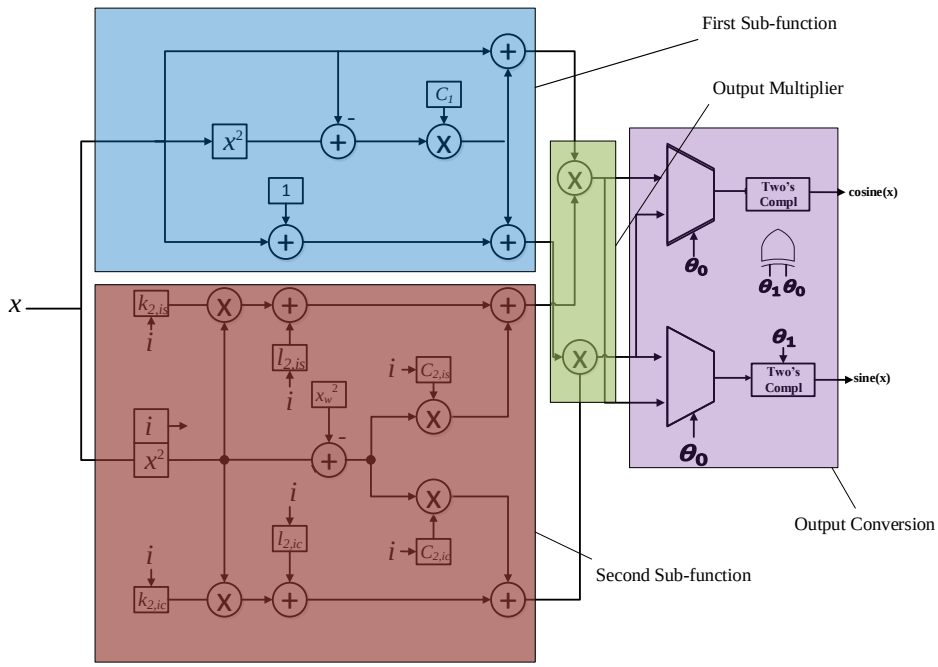


Figure 8.2: Different modules in the design

For the sake of comparison with previous work, we can approximate the area needed to calculate single function e.g., sine from the Table VIII. A rough calculation is given in Table IX.

TABLE IX: APPROXIMATED AREA FOR SINGLE FUNCTION

Module	Approximated area (μm²)
First Sub-function	2400
Second Sub-function	4224
Output Multipliers	2000
Two's Conversions	300
In/out DFFs	300
Total	9224

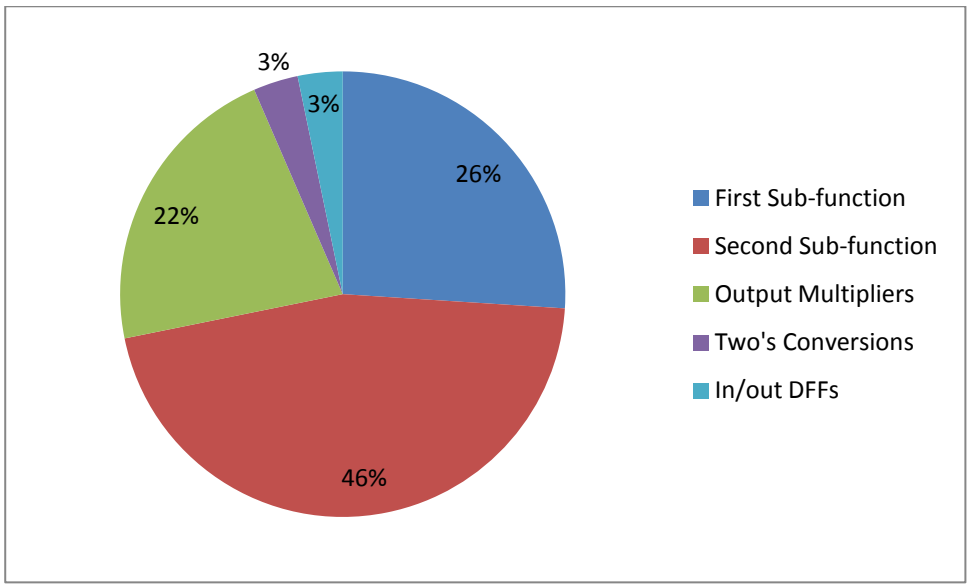


Figure 8.3: Approximated area for one function

8.1.2 Timing/Speed Results

The maximum speed of the design is calculated by setting the timing constraint in design vision to 1 ns. This gives an *x* value of negative slack

for the critical path. The *x* value is added to 1 ns and the simulation is performed again unless the slack is zero.

TABLE X: SPEED RESULTS FOR LPHVT AND LPLVT AT NORMAL CONSTRAINST

	LPHVT			LPLVT		
Voltage (V)	**1.00**	**1.10**	**1.20**	**1.00**	**1.10**	**1.20**
Speed (MHz)	30.91	42.51	52.96	73.52	86.50	100.20
Time (*ns*)	32.35	23.52	18.88	13.60	11.56	9.98

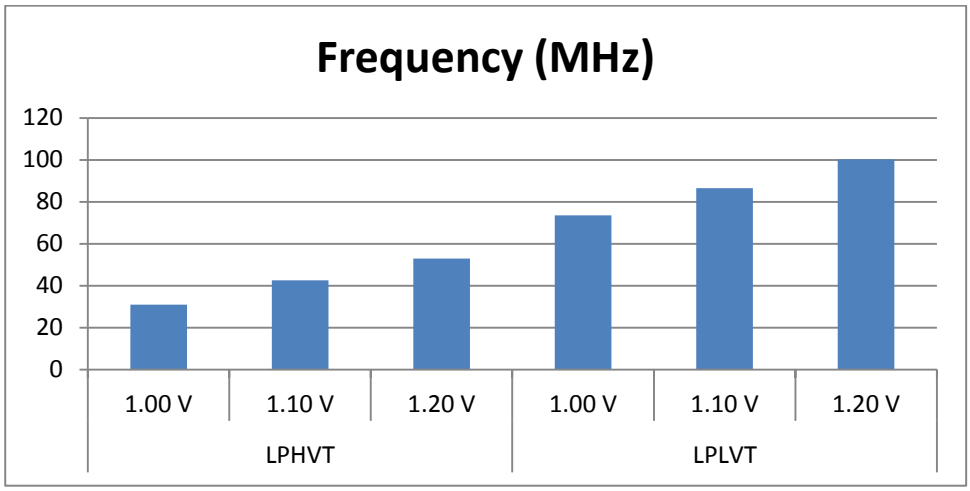


Figure 8.4: Frequency results for LPHVT and LPLVT

It can be seen that LPLVT transistors are considerably faster than LPHVT transistors. The maximum frequency using LPHVT transistor is 135.5 MHz at 1.20 volts of supply voltage where as in case of LPLVT transistors it is 265.25 MHz at the same voltage.

8.1.3 Power Results

The power dissipation in a CMOS transistor consists of two sources given by (8.1).

$$P_{total} = P_{dynamic} + P_{static} \quad (8.1)$$

The dynamic power is the total switching power and the internal power. It depends on the charging and discharging of the capacitances, switching activity, supplied voltage and the operating frequency as given in (8.2).

$$P_{dynamic} = \propto CV^2 f \quad (8.2)$$

where

$\propto$ = Switching activity
C = Capacitance
V = Supplied voltage
f = Clock frequency

The power consumption for the Parabolic Synthesis and Non-Linear Interpolation is simulated using the PrimeTime tool at a frequency of 10MHz at the supply voltages mentioned above. In order to analyze the power consumption of the design a Value Change Dump (VCD) file is generated in ModelSim using the netlist file generated during the synthesis process. The power results for both LPHVT and LPLVT are given below.

TABLE XI: POWER ANALYSIS USING LPHVT LIBRARIES AT DIFFERENT VOLTAGES

	LPHVT		
Voltage (V)	**1.00**	**1.10**	**1.20**
Net Switching Power (µW)	11.26	13.87	16.65
Cell Internal Power (µW)	12.04	14.74	17.63
Cell Leakage Power (*n*W)	23.35	33.5	48.96
Total Power (µW)	**23.32**	**28.64**	**34.33**

TABLE XII: POWER ANALYSIS USING LPLVT LIBRARIES AT DIFFERENT VOLTAGES

	LPLVT		
Voltage (V)	**1.00**	**1.10**	**1.20**
Net Switching Power (µW)	12.16	15.16	18.32
Cell Internal Power (µW)	13.11	17.05	21.95
Cell Leakage Power (µW)	4.20	6.54	9.83
Total Power (µW)	**29.47**	**38.75**	**50.10**

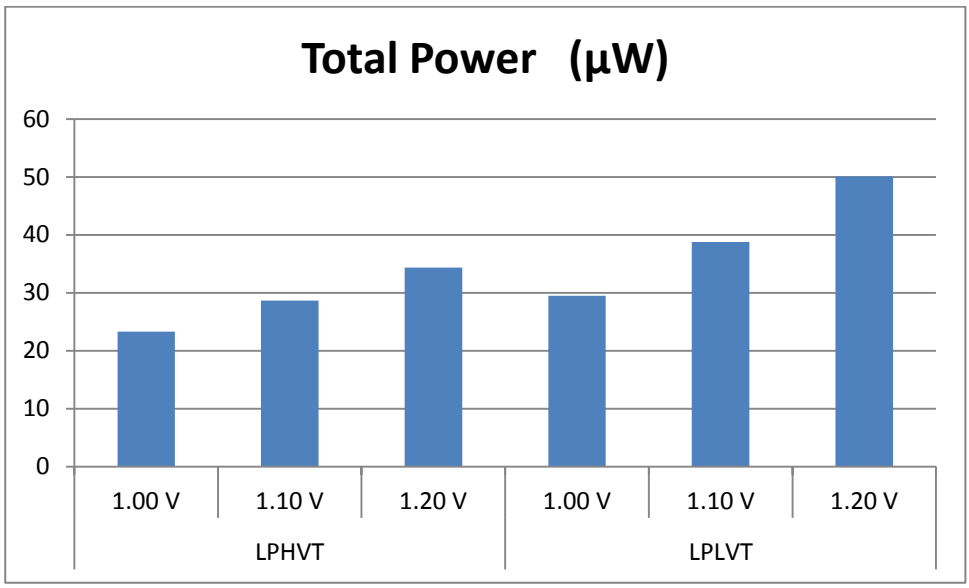


Figure 8.5: Total power comparison for LPHVT and LPLVT at the frequency 10MHz

The cell leakage power increases with increased supply voltage. It should be noted that static power dissipation (cell leakage power) is considerably high in LPLVT transistors as compared to LPHVT.

8.2 Existing Algorithms

In this section, the area, speed and power results of the Parabolic Synthesis combined with Non-Linear Interpolation are compared to other algorithms like the CORDIC and previous thesis work on Parabolic Synthesis methodology like 'Sine Function Approximation using Parabolic Synthesis and Linear Interpolation' [15] and "Hardware Implementation of Logarithm function using improved parabolic synthesis"[16]. However, it is not possible to compare the results precisely, since the above mentioned algorithms were implemented for different functions, e.g., sine or logarithm and the operating frequency of the design to find the power dissipation is not mentioned clearly.

The implementation results compared in this section are taken from the thesis work sine function implementation by Madhubabu Nimmagadda and Surendra Reddy Utukuru[15], Improved Parabolic Synthesis by Jingou Lai[16] and Logarithmic and exponential function implementation by Peyman Pouyan[8]. As mentioned before that the Parabolic Synthesis methodology can be used to implement different unary functions using the same architecture with different set of coefficients. Hence it is possible to compare the results of different implementations. However, the CORDIC algorithm has a simple hardware to implement the trigonometric and logarithmic functions. It is implemented by using simple shift and add operations and a look-up table (LUT). In order to get a precision of 15 bits, more than 15 iterations will be required, which will increase its computation time considerably. However, almost the same resolution is achieved in this thesis work by combining Parabolic Synthesis with Non-Linear Interpolation.

The chip area result for different methodologies is given in the Table XIII.

TABLE XIII: AREA ANALYSIS OF ASIC IMPLEMENTATION FOR DIFFERENT METHODOLGIES(LPHVT @ 1.20V)

Methodology	Area (μm^2)
CORDIC	19048 [1]
Parabolic Synthesis	25249 [1]
Parabolic Synthesis with Linear Interpolation	11397 [2]
Parabolic Synthesis with Non-Linear Interpolation	15982
Improved Parabolic Synthesis	5894

[1] The results are with pads

[2] The analysis is done at 1.25 volts

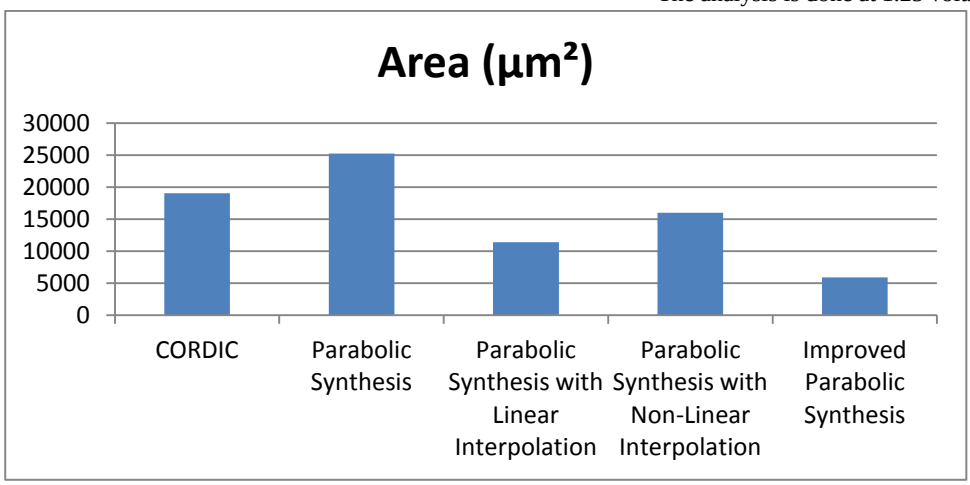


Figure 8.6: ASIC synthesis analysis for area

The parabolic synthesis combined with non-linear interpolation occupies less area compared to the CORDIC and it can be used to implement different unary functions like the logarithmic, exponential, division and square-root function. Only the set of coefficients in the look-up table (LUT) are needed to be changed to implement a different unary function with the main architecture unchanged [8]. On the other hand, the CORDIC algorithm needs a different architecture and extra iterations in order to implement logarithmic function.

TABLE XIV: FREQUENCY FOR THE ASIC IMPLEMENTATION OF DIFFERENT METHODOLOGIES (LPHVT @ 1.20V)

Methodology	Frequency (MHz)	Critical Path Delay (*ns*)
CORDIC	11.5	86.72
Parabolic Synthesis	47.5	21.47
Parabolic Synthesis with Linear Interpolation	58.82 [3]	18.18 [3]
Parabolic Synthesis with Non-Linear Interpolation	53.99	18.52
Improved Parabolic Synthesis	83.33	12.00

[3] The analysis is done at 1.15 volts

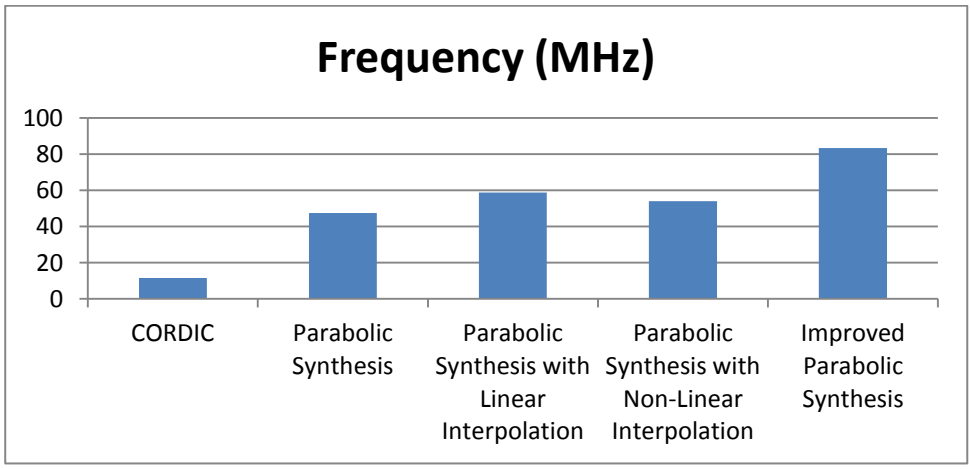


Figure 8.7: Frequency for the ASIC implementation of different methodologies

The ASIC implementation shows that the parabolic Synthesis combined with non-linear interpolation is 4.6 times faster than the CORDIC, 1.16 times faster than the non-pipelined Parabolic Synthesis. It should be noted that this design uses an extra multiplexer and two's complement conversion at the output for output transforms, which adds up to increase the critical

path. These results are compared for LPHVT implementation at a supply voltage of 1.20 Volts.

Table XV shows the estimated power consumption for different methodologies with a LPHVT implementation at a frequency of 10 MHz using the PrimeTime tool.

TABLE XV: POWER ANALYSIS FOR DIFFERENT METHODOLOGIES (LPHVT @1.20V)

Methodology	**Power @ 10 MHz (µW)**
CORDIC	99.79
Parabolic Synthesis	98.08
Parabolic Synthesis with Linear Interpolation	39.4(@1.25V)
Parabolic Synthesis with Non-Linear Interpolation	34.33
Improved Parabolic Synthesis	17.38

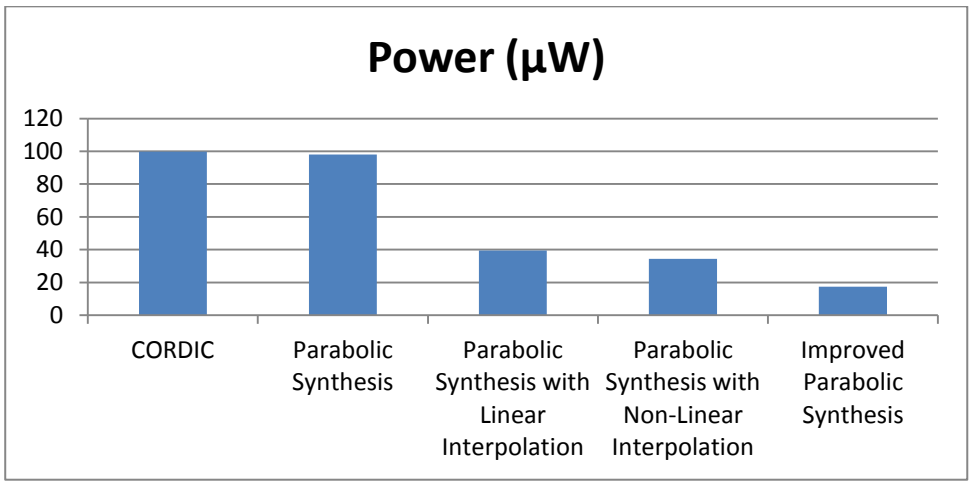


Figure 8.8: Power dissipation analysis for different algorithms

Since the design occupies less area and there is a lower switching activity the Parabolic Synthesis combined with Non-Linear Interpolation therefore

uses less power as compared with CORDIC and Parabolic Synthesis. It is not possible to fairly compare the power results since the other algorithms were implemented to approximate only one function however, in this design is used to implement two trigonometric functions i.e. sine and cosine at the same time.

CHAPTER 9

9 Conclusions

The approximation of the trigonometric identities i.e., sine and cosine was designed and implemented in the same architecture using Parabolic Synthesis combined with Non-Linear Interpolation. Four intervals were used in the interpolation part which leads to a set of 12 coefficients for both functions i.e., sine and cosine.

The truncation changes the error behavior. In order to compensate for that, the coefficients need to be optimized manually. This should be done by changing one set of coefficients at a time.

The architecture was carefully designed to have high degree of parallelism therefore it has short critical path and fast computation speed. The design is suitable for high speed applications since it is much faster than the CORDIC and other implementations for the same resolution.

Certain simplifications were done in the architecture that includes designing a special squarer to find x^2 and x_w^2 using the same architecture, which makes the area less as compared to the Parabolic Synthesis with Linear Interpolation.

The resolution of the approximation is almost 15 bits, which is according to the required resolution for this thesis work. The error behavior indicates that the error of approximation is evenly distributed around zero.

CHAPTER 10

10 Future Work

The error behavior can be improved by using more intervals in the second sub-function, $s_2(x)$. It will increase the look-up table (LUT) size but there can be a compromise between area and accuracy.

The architecture of the approximation can be made faster by introducing pipelining at different stages.

The same architecture can be used to calculate different unary functions including various trigonometric, logarithmic, exponential, division and square-root function by only changing the set of coefficients used.

References

[1] P. T. P. Tang, “Table-lookup algorithms for elementary functions and their error analysis,” in *Proc. of the 10th IEEE Symposium on Computer Arithmetic ISBN: 0-8186-9151-4*, pp. 232 - 236, Grenoble, France, June 1991.

[2] J. M. Muller, “Elementary Functions: Algorithm Implementation,” in *second edition Birkhauser, ISBN 0-8176-4372-9*, Birkhauser Boston, c/o Springer Science+Business Media Inc., 233 Spring Street, New York, NY 10013, USA.

[3] Ateeq Ur Rahman Shaik, “Hardware Implementation of the exponential function using Taylor series and Linear Interpolation”, Lund University Mater Thesis, April 2014.

[4] Erik Hertz, “Parabolic Synthesis”, Thesis for the degree of Licentiate in Engineering, Lund University, 2011.

[5] Ray Andraka, “A survey of CORDIC algorithms in FPGA based computers”, Andraka Consulting Group, Inc. North Kingstown, USA.

[6] Muhammad Waqas Shafiq and Nauman Hafeez, “Design of FFTs using CORDIC and Parabolic Synthesis as an alternative to Twiddle Factor Rotations”, Lund University, Master Thesis, 2011.

[7] Erik Hertz , Bertil Svensson, and Peter Nilsson, “Combining the Parabolic Synthesis Methodology with Second-Degree Interpolation”. Centre for Research on Embedded Systems, Halmstad University, Halmstad, Sweden, Electrical and Information Technology Department, Lund University, Lund, Sweden.

[8] Peyman Pouyan, Erik Hertz, and Peter Nilsson, “A VLSI Implementation of Logarithmic and Exponential Functions Using a Novel

Parabolic Synthesis Methodology Compared to the CORDIC Algorithm", 20th European Conference on Circuit Theory and Design (ECCTD), 2011.

[9] E. Hertz and P. Nilsson, "A Methodology for Parabolic Synthesis," a book chapter in *VLSI, In Tech, ISBN 978-3- 902613-50-9*, pp. 199-220, Vienna, Austria, September 2009.

[10] E. Hertz and P. Nilsson, "Parabolic Synthesis Methodology Implemented on the Sine Function," in *Proc. of the 2009 International Symposium on Circuits and Systems (ISCAS'09)*, Taipei, Taiwan, May 24-27, 2009.

[11] Parabolic Synthesis. http://www.intechopen.com/articles/show/title/a-methodology-for-parabolic-synthesis

[12] J.-M. Muller, Elementary Functions: Algorithm Implementation, second ed. Birkhauser, ISBN 0-8176-4372-9, Birkhauser Boston, c/o Springer Science+Business Media Inc., 233 Spring Street, New York, NY 10013, USA, 2006.

[13] A. A. Giunta and L. T. Watson, "A Comparison of Approximation Modeling Techniques," *American Institute of Aeronautics and Astronautics*, AIAA-98-4758, Blacsburg, USA, September 1998.

[14] Personal discussion and helping material provided by Peter Nilsson and Erik Hertz.

[15] Madhubabu Nimmagadda and Surendra Reddy Utukuru, "Sine Function Approximation using Parabolic Synthesis and Linear Interpolation", Master Thesis, Lund University,2011.

[16] Jingou Lai, "Hardware Implementation of the Logarithm Function using Improved Parabolic Synthesis", Master Thesis, Lund University, 2013.

List of Figures

List of Tables

List of Acronyms

CMOS	Complementary Metal Oxide Semiconductor
DSP	Digital Signal Processing
CORDIC	COordinate Rotation DIgital Computer
VHDL	VHSIC Hardware Design Language
VHSIC	Very High Speed Integrated Circuit
RMS	Root Mean Square
MUX	Multiplexer
AC	Alternating Current
DC	Direct Current
MSB	Most Significant Bit
LPLVT	Low Power Low V_t
LPHVT	Low Power High V_t
VCD	Value Change Dump
LUT	Look-Up Table

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2015-422

http://www.eit.lth.se