Master's Thesis

Sine Function Approximation using Parabolic Synthesis and Linear Interpolation

By

Madhubabu Nimmagadda
Surendra Reddy Utukuru

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

Abstract

This Thesis work deals with the ASIC implementation of the sine function approximation using the combination of Parabolic Synthesis methodology and linear interpolation. Parabolic Synthesis is a novel methodology for implementing unary functions such as logarithmic, trigonometric, and arithmetic functions. This methodology gives a high degree of parallelism and efficient use of hardware compared to existing algorithms like polynomial approximation, CORDIC etc. Parabolic Synthesis is combined with linear interpolation in order to achieve the high accuracy. The architecture is designed to achieve increased performance, lesser area and lower power consumption.

Keywords: Parabolic Synthesis, Linear interpolation, MCM, Sine function

Acknowledgments

It is a pleasure to thank everyone who has made this dissertation possible and helped us throughout the project with their excellent guidance. We would not have finished our thesis without their support and motivation.

First, we would like to express our gratitude to Professor Peter Nilsson, who has provided the thesis and let us experience in the field of Digital ASIC. We have had good guidance, patience, caring, and very pleasant work atmosphere.

Second, our deepest thanks to our supervisor Erik Hertz, who had given the best suggestions and solutions to get out of the problems faced during our entire project. He had given us his valuable assistance and appreciation toward the work we have done. We will never forget his smile and his friendly nature toward explaining, making doubts clear. We are thankful to the Department of EIT, who has provided and given the access to the most expensive project tools.

Finally, we would like to thank our friends and parents, who always support us in our life and keep up our happiness.

Madhubabu
NimmagaddaSurendra Reddy
Utukuru

Table of Contents

Preface

In this thesis work, we both have worked together except from the literature work, and total work has been scheduled based on our availability. We have divided the tasks and worked together throughout the project.

This thesis documentation is organized into 8 chapters. Chapter 1 presents a short introduction to exist algorithms along with their merits and demerits. Chapter 2 provides description of Parabolic Synthesis methodology and its advantages over other algorithms. Chapter 3 explains the linear interpolation technique and chapter 4 contains a detailed description of the architecture development procedure for the parabolic synthesis methodology and linear interpolation approximation. Architecture implementation of design in hardware is explained in chapter 5. In chapter 6, Design flow and results from the hardware implementation are discussed and compared with other implementations. Conclusion and future work is explained in chapter 7 and 8.

CHAPTER 1

1 Introduction

Unary functions like Logarithmic, Exponential, and Trigonometric functions as well as Arithmetic functions [1] plays an important role in applications like wireless systems, computer graphics, digital signal processing. For the implementation of these functions we have to choose an algorithm, which is good and plays an important role in area, computational speed, and accuracy. The hardware implementation of such functions is a challenge to meet the desired requirements.

There exist different algorithms such as single lookup table, polynomial approximation, and CORDIC algorithm, which has own merits and demerits. A single lookup table [2] is an easy and direct method in implementation. However this method suits for the low precision applications, not for low area applications. A Single lookup table can be approximated by using interpolation technique in order to increase the accuracy up to an extent.

Polynomial approximations [3] are performed with additions and multiplications. The computational complexity will be depending on which polynomial schemes you will choose in the existed schemes. Depending on the requirements of the design, polynomial approximation can be implemented either with least squares approximation or with least maximum approximation.

Another method is the CORDIC algorithm [4], which is good in efficient use of hardware since it has no multipliers. The CORDIC algorithm is an iterative algorithm, which is implemented by using shifts and additions. Increased number of iterations increases the bit accuracy but makes the algorithm unfeasible for high speed applications.

In order to overcome some limitations in the previous methods, a new algorithm called Parabolic Synthesis [5] has been chosen. It is a novel methodology that came up with a very fast computational techniques and better accuracy, when implementing in hardware. In this Master thesis work, the design of the sine function approximation is done by using Parabolic Synthesis methodology and

followed by linear interpolation approximation aiming for better accuracy, area, and power consumption.

CHAPTER 2

2 Parabolic Synthesis

2.1 Introduction

Parabolic Synthesis is a different methodology to implement functions like Trigonometric, Logarithmic, and Arithmetic functions, which plays a vital role in the field of signal processing, image processing, and robotics, where the speed of computations has an important role [5].

The Parabolic Synthesis methodology makes use of an original function to develop the sub functions and help functions, which are implemented in the hardware [5]. The proposed architecture possesses a high level of parallelism and consumes less time in computation. The design is built only with multipliers, adders and shifters, which is easily implemented in hardware.

The Parabolic Synthesis methodology works better in getting efficient response in terms of area, power and time. The main steps in this methodology are normalization, approximation and transformation [5].

2.2 Normalization

Normalization is the first step in Parabolic Synthesis methodology, which limits the input in the numerical range 0 to 1.The inputs and output for a particular unary function ,which is notated as $f_{org}(x)$,should be limited to 0 to 1 on both x and y axis. The function should be either convex or concave throughout the interval.

2.3 Methodology development

The Parabolic Synthesis methodology is developed in approximation of any unary function, it always good to think in hardware perspective so that they can be synthesized with good metrics. The simple operations like multiplication, addition, and shifts are chosen in designing architecture.
The proposed methodology will be derived in to a set of sub functions $s_n(x)$, which are recombined in order to get an approximated original function.

$$\boldsymbol{f_{org}(x) = s_1(x) * s_2(x) \ldots . s_n(x)} \quad (\mathbf{2.1})$$

In (2.1), $s_1(x)$ represents a parabolic looking function, which is called first help function. The use of number of sub functions closes the approximated value to original value and hence better accuracy. The sub functions derived as parabolic looking functions should be feasible, so that they can be implemented in hardware. The second sub function is developed by using first help function $f_{diff}(x)$.

$$f_{diff}(x) = \frac{f_{org}(x)}{s_1(x)} \quad (2.2)$$

In the same manner, the other help functions are also developed in order to develop other sub functions as shown in (2.3).

$$f_{n+1}(x) = \frac{f_n(x)}{s_{n+1}(x)} \quad (2.3)$$

Developing sub-functions

The first sub functions$s_1(x)$ in Parabolic Synthesis methodology is developed by dividing the original function$f_{org}(x)$, with *x* as first order approximation. The two possibilities after the division is either$f(x)$> 1 or $f(x)$< 1.To develop the first sub function,$1 + \big(c_1(1-x)\big)$ is used an approximated term. The multiplication of the approximated term with *x* gives the first sub-function [1] [16].

$$s_1(x) = 1 + \big(c_1(1-x)\big) * x = x + c_1(x - x^2) \quad (2.4)$$

The coefficientc1 in (2.4) is calculated determined according to (2.5)

$$c_1 = \lim_{x \to 0} \frac{f_{org}(x)}{x} - 1 \quad (2.5)$$

The first help function is calculated according to (2.2) and is used in developing the second sub function, which is also a parabolic function.

$$s_2(x) = 1 + c_2(x - x^2) \tag{2.6}$$

c_2 in (2.6) is chosen to satisfy that the quotient between the first help function $f_{help1}(x)$ and the second sub-function $s_2(x)$ is equal to 1 when x is set to 0.5.

$$c_2 = 4\left(f_1\left(\frac{1}{2}\right) - 1\right) \tag{2.7}$$

The sub functions $n > 2$ are developed by dividing the help function into sub intervals as equated in (2.8). The parts of help functions are used to develop further partial sub functions [1].

$$f_n(x) = \begin{cases} f_{n,0}(x), 0 \le x \le \frac{1}{2^{n-1}} \\ f_{n,1}(x), \frac{1}{2^{n-1}} \le x \le \frac{2}{2^{n-1}} \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots \\ f_{n,2^{n-1}}(x), \frac{2^{n-1}-1}{2^{n-1}} \le x \le 1 \end{cases} \tag{2.8}$$

The normalized value of input x_n is used instead of x in partial sub functions in order to have a feasibility to implement them in hardware. The partial sub functions are notated as (2.9) [1] [16].

$$s_n(x) = \begin{cases} s_{n,0}(x_n), 0 \le x \le \frac{1}{2^{n-2}} \\ s_{n,1}(x_n), \frac{1}{2^{n-2}} \le x \le \frac{2}{2^{n-2}} \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots \\ s_{n,2^{n-2}}(x_n), \frac{2^{n-2}-1}{2^{n-2}} \le x \le 1 \end{cases} \tag{2.9}$$

The recombination of partial sub functions in (2.9) gives an approximation of $f_{n-1}(x)$ and parabolic sub function is formulated as (2.10)[1] [16].

$$s_{n,m}(x) = 1 + c_{n,m} * (x_n - x_n^2) \tag{2.10}$$

Coefficients in parabolic sub functions are calculated according to (2.11)

$$c_{n,m} = 4 * \left\{ f_{n-1,m} * \left(\frac{2 * (m+1) - 1}{2^{n-1}} \right) - 1 \right\} \tag{2.11}$$

In parabolic synthesis methodology, to implement any unary function using more than two sub functions results more hardware. The third sub function uses two coefficients and needs one more multiplier to recombine, which is a demerit in hardware perception. It is always a tradeoff to choose how many bits of accuracy are desired and how many number of sub functions should be used.

CHAPTER 3

3 Interpolation

3.1 Introduction

The interpolation concept is used to approximate the function with the known data points by analyzing the numerical data [9]. There are several ways of interpolating a continuous signal like Linear, Piecewise, and spline. From all the existing interpolation techniques, linear interpolation is having its own advantages which possess speed, simplicity, and accuracy [9].

3.2 Interpolation methods

There are several interpolation methods existed and every method is having its own merits and demerits. Depends on application and implementation, user has to make a tradeoff to choose which method is best suited. Piecewise constant interpolation, Linear interpolation, Spline, and the Polynomial interpolation are different interpolation methods, which are commonly used in approximation of functions.

Piecewise constant interpolation or nearest neighbor interpolation is the easiest and the simplest method among all the existing methods [10]. In this technique, the method just locates the nearest data value and assigns to it. This method is chosen for image processing applications where multivariate interpolation is needed [9].

Linear interpolation is the other method, which uses some known data points to calculate an unknown data point [9]. This method is well suited for the applications like computer graphics and easily implemented with Lookup tables. To work in other spatial domains, one can go for bilinear interpolation or trilinear interpolation methods.

Polynomial interpolation is another form of interpolation, where the interpolant is a polynomial function, which is a linear equation in case of linear interpolation [11]. Here the Interpolation error is proportional to the distance between the data points and interpolant is indefinitely differentiable, since it is a polynomial [11]. On the other hand, the polynomial interpolation has a drawback of computational complexity and oscillatory artifacts [11]. The spline interpolation method

overcomes the disadvantages of polynomial method by using a low degree of polynomial. The computational complexity is also lower with same accuracy as the polynomial interpolation [11].

3.3 Linear interpolation

Linear interpolation is a method of finding new data points *(x, y)* from the known data points $(x_0\ y_0,\ x_1 y_1)$ as shown in Fig. 1. The new set of data points *(x, y)* can be found by the slope of straight line between two known points [9].

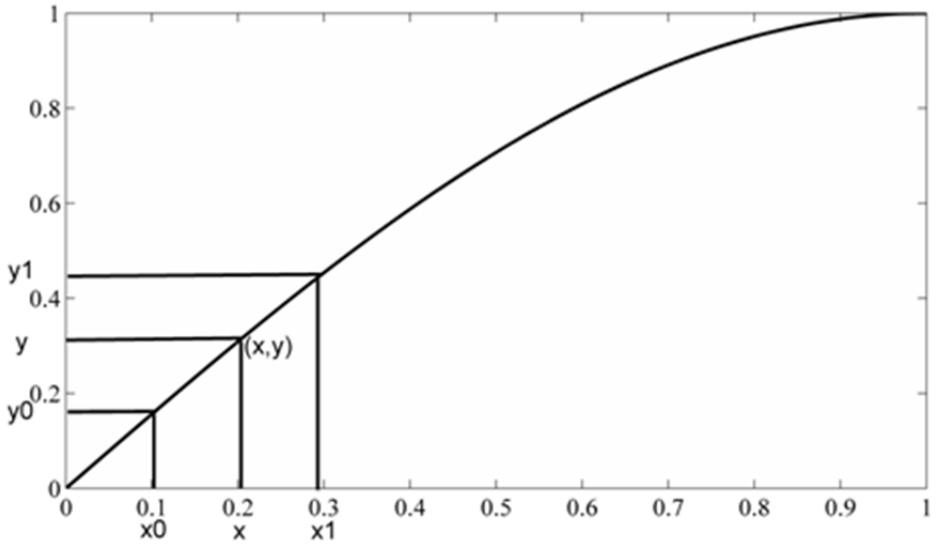

Figure 1. The Method of interpolation

As shown in Fig. 1, the value of y for a known value of x is calculated according to (3.1).

$$y = y_0 + (x - x_0) * \frac{y_1 - y_0}{x_1 - x_0} \tag{3.1}$$

Elaborating (3.1), y_0 is the value from where the interpolation starts and the second part in the product term is the gradient value of the line. Linear interpolation with different intervals like 8, 16, 32 and 64 are chosen to get desired accuracy and precision of values.

CHAPTER 4

4 Architecture development

4.1 Introduction

In this section, architecture development for sine function using Parabolic Synthesis methodology combined with linear interpolation is explained. As explained in section 2, Parabolic Synthesis methodology provides feasibility in implementation of unary functions in accuracy and hardware utilization also. Providing some optimizations for architecture also makes more efficient hardware implementation. In this case, sine function is the original function f_{org} (x) and is used to develop the sub functions. There are several algorithms to implement a sine wave in hardware as explained in section 1, every algorithm is having its own merits and demerits. In this project, a novel methodology Parabolic Synthesis is chosen to gain the less computation complexity and speed as well. Parabolic Synthesis is a methodology, which is constructed with different stages called as sub functions to compute value for any unary function [5]. The Error from these sub functions is analyzed and approximated by using linear interpolation technique.

An analysis is made to choose the number of sub functions to be used in achieving more than 15 bits of accuracy. The third sub function is implemented with linear interpolation technique, which gives a better result in all the metrics. The objective of the design is achieving better accuracy with certain input constraints, less power consumption and less computational speed.

4.2 Analysis

The Parabolic synthesis architecture consists of different sub functions and the product of them gives an approximated original function. The accuracy of the output increases as the number of sub functions is increased. Increasing the number of sub functions consumes more hardware area and computational speed. So, an analysis is made for architecture to use how many sub functions are to be combined with linear interpolation in order to achieve the accuracy desired. The accuracy is calculated by measuring the difference between original function and derived function.

$$absolute\ error(dB) = 20 * \log(abs(original\ value - derived\ result)) \quad (4.1)$$

In dB scale 6.02 dB represents 1 bit accuracy. For example, 61 dB represents 10 bit accuracy. In this case, two different scenarios are considered to analyze the absolute error and accuracy, which are discussed further.

4.2.1 One sub-function combined with linear interpolation

The sine function approximation is designed with one sub function combined with different levels of interpolation to analyze the bit accuracy. The design is approximated in the range $0^0<x<90^0$and 8, 16, 32, and 64 levels of interpolation is used.

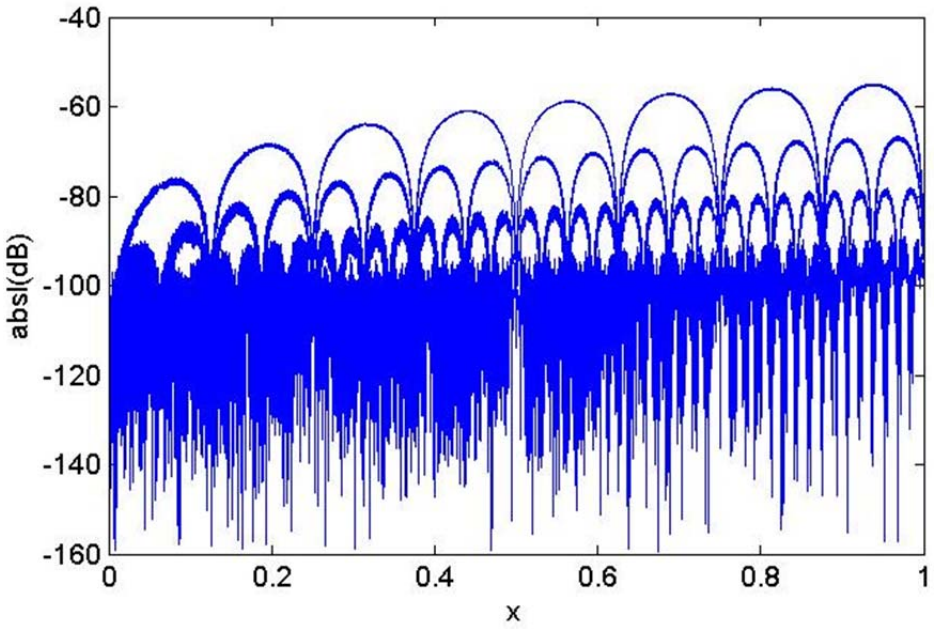

Figure 2. The Error analysis for 1 sub function & linear interpolation

In Fig. 2, the error analysis is shown for different scenario in dB value. The resolution in bits is as below.

- 8 intervals, resolution 9.1 bits
- 16 intervals, resolution 11.1 bits
- 32 intervals, resolution 13.09 bits
- 64 intervals, resolution 15.1 bits

As the number of interpolation levels is doubled, the resolution is increased by 2 bits. In order to achieve 15 bit accuracy, a 64 level interpolation is needed and in order to implement in hardware, a Look up Table (LUT), which can store 64 different values should be used.

4.2.2 Two sub-function combined with linear interpolation

As explained in 1.2.1, in order to achieve more than 15 bit accuracy, it requires more hardware area with one sub function combined with linear interpolation. In this section, the sine function is approximated by using the 2 sub functions combined with linear interpolation. The design is approximated in the range $0^0<x<90^0$and 8, 16, 32, and 64 levels of interpolation is used.

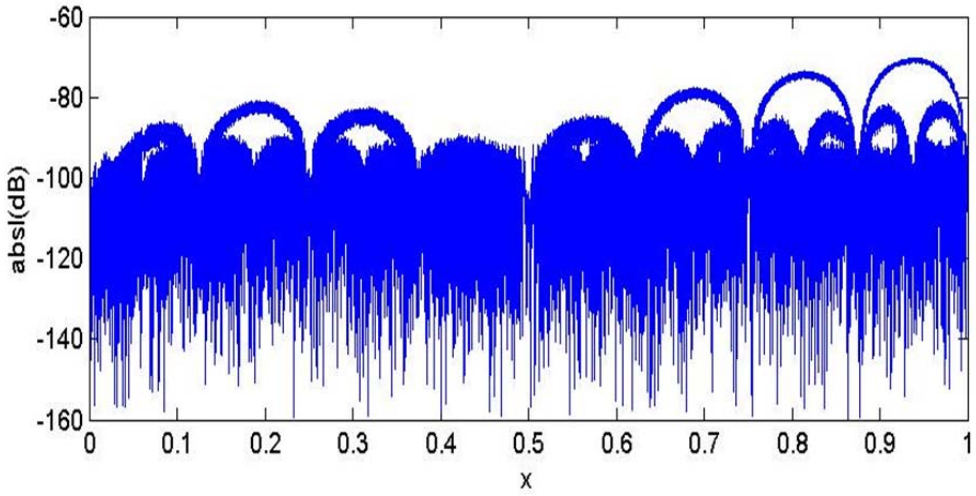

Figure 3. The Error analysis for 2 sub functions & linear interpolation

In Fig. 3, the error analysis for sine function approximation with 2 stage parabolic synthesis combined with linear interpolation is shown in dB scale. The resolution for different levels of interpolation is as below.

- 8 intervals, resolution 11.7 bits
- 16 intervals, resolution 13.4 bits

- 32 intervals, resolution 15.6 bits
- 64 intervals, resolution 17.5 bits

The sine function approximation designed with 2 stage parabolic synthesis combined with 32 level interpolation gives more than 15 bits of accuracy.

4.2.3 Comments

The approximation of the sine function implemented with parabolic synthesis methodology provides feasibility to implement in hardware. The linear interpolation approximation technique when combined to Parabolic Synthesis provides good results in accuracy, chip area, and computational speed as well. As the number of levels of interpolation is doubled, the resolution and error distribution is increased by 2 bits. Changing the coefficients, in linear interpolation manually, can increase the bit accuracy. The implementation of linear interpolation methodology for the approximation of sine function requires memories and more access time. Therefore, the concept of a 2-stage Parabolic Synthesis combined with linear interpolation is the feasible way to achieve less size memories and less access time.

4.3 Development of architecture

4.3.1 Parabolic Synthesis architecture

The Parabolic Synthesis methodology makes use of an original function to develop the sub functions and help functions, which are implemented in the hardware [5]. The proposed architecture possesses a high level of parallelism and consumes less time in computation. The design is built only with multipliers, adders and shifters, which is easily implemented in hardware.

The Parabolic Synthesis methodology works better in getting efficient response in terms of area, power and time. The main steps in this methodology are normalization, approximation and transformation [5].

4.3.1.1 Normalization

The Parabolic Synthesis methodology for a specific unary function can be computed mainly in three steps, normalization, approximation, and transforming.
In Normalization, the numerical range will be limited to a range in order to simplify the hardware implementation. For any Unary function $y=f_{org}(x)$, both x axis and y axis should be in range of 0 and 1. Fig. 4 shows sine function after normalization.

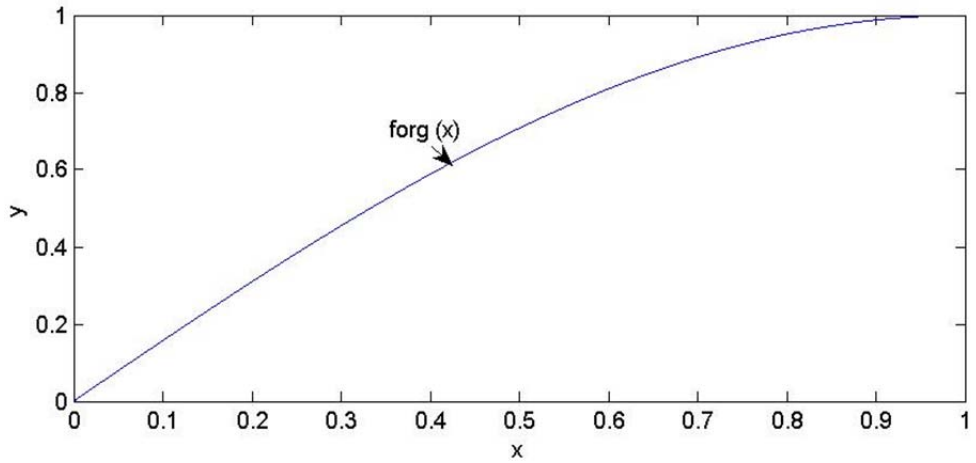

Figure 4. The Normalization for Sine function

4.3.1.2 Calculation of coefficient in sub-functions

The hardware is completely designed with low complexity operations like multiplications, additions and shifts when approximating the original function. The proposed methodology uses the basic function to formulate the sub functions and later multiplication of the sub functions gives the approximated original function.

The methodology is constructed with second-order parabolic functions [5] named as sub functions and recombination of these sub functions gives approximation of original function $f_{org}(x)$, in this case the sine function, as shown in (2.1). The accuracy will be better, if more sub functions are used in the calculation.

The help function generated at each stage is used in calculating the next sub function. For example as shown in (2.2), the division of the original function by the first sub function gives the first help function, which is used when deriving the second sub function. The subsequent help functions are generated from the according to (2.3).

When developing the sub functions, the original function is decomposed into second-order parabolic functions for the interval 0 to 1. The second order parabolic functions are chosen to avail a benefit of implementing with less complexity in hardware. The third sub function from the Parabolic Synthesis methodology contains two coefficients which increases the hardware complexity [9]. The accuracy will be better by replacing the third sub function with another approximation technique.

The first sub function is calculated by dividing the original function with x as a first order approximation, which is specified in (2.2). The result have two possibilities, $f_{diff}(x)>1$ and $f_{diff}(x)<1$ [14]. In order to approximate the original function, $f_2(x)$ is developed from $f_{diff}(x)$, which is specified in (2.3). The first sub function is calculated by multiplying x and $f_2(x)$ as shown in (4.3). The related plot in developing first sub function is shown in Fig.5.

$$c_1 = \lim_{x \to 0} \frac{f_{org}(x)}{x} - 1 \quad = 0.5703125 \tag{4.2}$$

$$f_2(x) = 1 + c_1(1 - x) \tag{4.3}$$

where c_1 is specified in (4.2)

$$s_1(x) = f_2(x) * x = x + c_1(x - x^2) \tag{4.4}$$

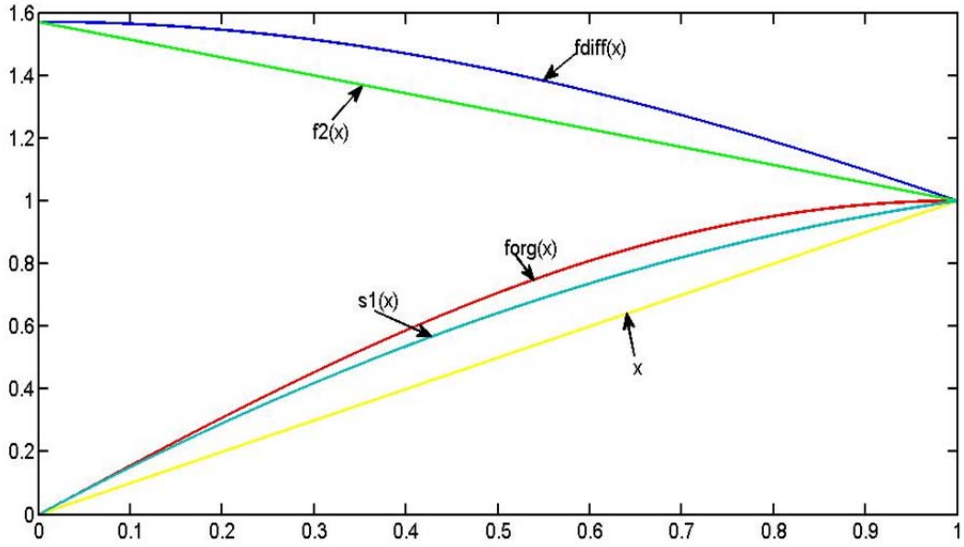

Figure 5. The Developing the first sub function

The first help function, calculated from (4.5) is used when deriving the second sub function. The second sub function $s_2(x)$ is calculated from (2.6).

$$f_{help1}(x) = \frac{f_{org}(x)}{s_1(x)} \tag{4.5}$$

c_2 in (2.6) is chosen to satisfy that the quotient between the first help function $f_{help1}(x)$ and the second sub-function $s_2(x)$ is equal to 1 when x is set to 0.5.

$$c_2 = 4\left(f_1\left(\frac{1}{2}\right) - 1\right) \quad = 0.400390625 \tag{4.6}$$

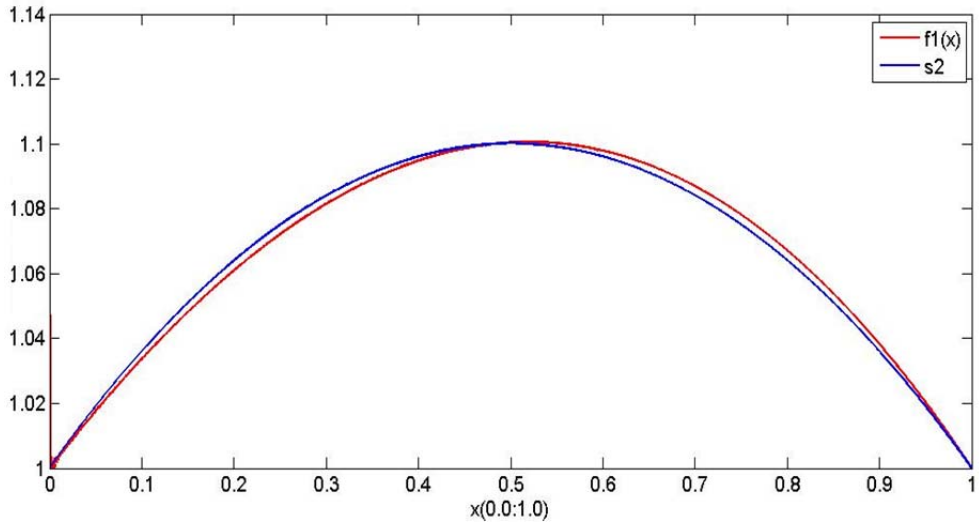

Figure 6. fhelp (x) & second sub function s2(x)

Fig. 6 shows plots for the help function and the second sub function. The crossing at *x=0.5* can be clearly seen from the graph.

4.3.2 Linear interpolation architecture

In case of the sine wave approximation, linear interpolation is applied by calculating the gradient values at different levels. Equation (4.7) gives the third sub function, which is used for the linear interpolation technique.

$$s_3(x) = K_{MSB} + (G_{MSB} * X_{LSB}) \tag{4.7}$$

From the equation, *K* is the value from where the interpolation is starting and *G* is the slope of the function. The most significant bits from the input *x* are used to index the particular gradient value and *K* value.

The values of *K* and *G* are measured from the help function generated by dividing the original function and result from Parabolic Synthesis as shown in (4.8).

$$Help\ function = \frac{f_{org}}{f_{parabolic}} \tag{4.8}$$

The value of *K* is derived from the help function and depends on the number of levels in the interpolation. For *n* levels of interpolation, *n* number of *K* values is chosen from the help function at successive time intervals. The gradient *(G)* value is calculated by subtracting the successive *K* values according to (4.9).

$$G(n) = K(n+1) - K(n) \tag{4.9}$$

Fig. 7 shows the plot of the *K* values measured from the help function.

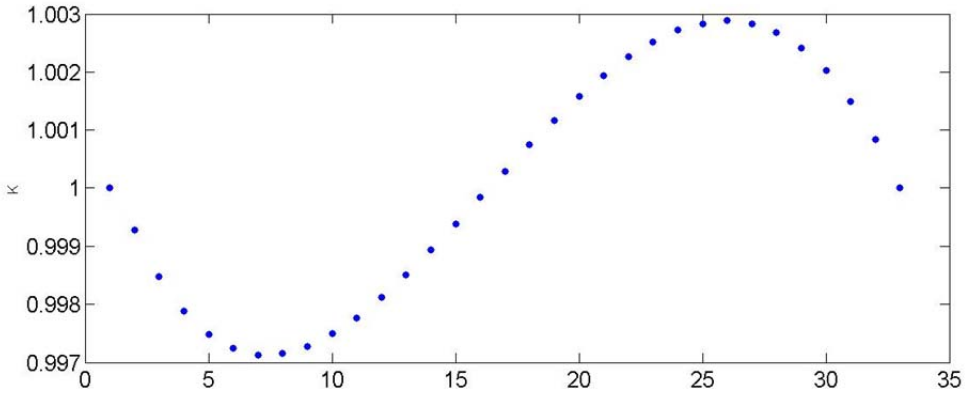

Figure 7. K values for the third sub function

4.3.3 Four quadrants

The calculations are developed for the first quadrant and extended to calculate the sine values in other three quadrants also. The original angle is normalized and transformed into first quadrant by using a factor *2/ π.*

$$Normalized\ angle(\theta_f) = \theta * \left(\frac{2}{pi}\right) = (q_1 q_0)(fractional\ part) \tag{4.10}$$

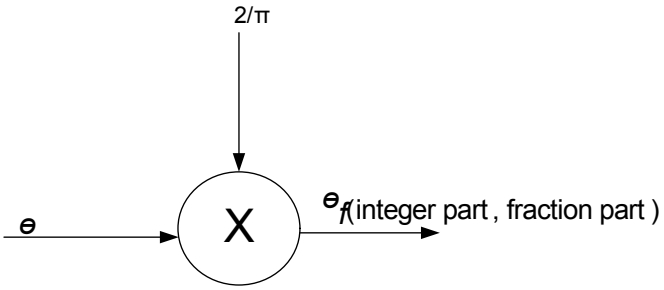

Figure 8.Normalization of original angle

For example, 2π/3 will be represented as

$$\frac{2\pi}{3}\ X\ \frac{2}{\pi} = \frac{4}{3} = 01\frac{1}{3} \tag{4.11}$$

The result in (4.11) is divided into integer part and fractional part. The integer part is represented using two binary bits q_1q_0 and is used by the multiplexer in the architecture. These two select lines represent the quadrant of the original angle. The same bits are also used in two's complement representation of the output when sine is in 3rd and 4th quadrant. The changes in architecture in view of the hardware are explained briefly in next chapter. The architecture for all four quadrants is possible by inserting one multiplexer and a two's complement converter.

Table 1. Input transformation for 4 quadrants

Quadrant	Theta_sel(q_1q_0)	Input value
1	00	θ_f
2	01	$1-\theta_f$
3	10	θ_f
4	11	$1-\theta_f$

The table 1 represents the q_1q_0 values for all the quadrants and input vales, which are selected by multiplexer.

CHAPTER 5

5 Hardware implementation and optimization

5.1 Design flow

The implementation of the design in hardware starts with making a reference model in MATLAB and verifying the functional behavior of the design. This thesis is organized similar to the ASIC flow as shown in Fig. 9. After a successfully working reference model, the design is coded in VHDL and simulated with the Modelsim tool. Furthermore, the results are compared with reference model. The design is synthesized for generating a netlist and is used to analyze the power consumption. All possible switching activities are generated as Value Change Dump (VCD) files using Modelsim, which is used by the Primetime tool for power analysis.

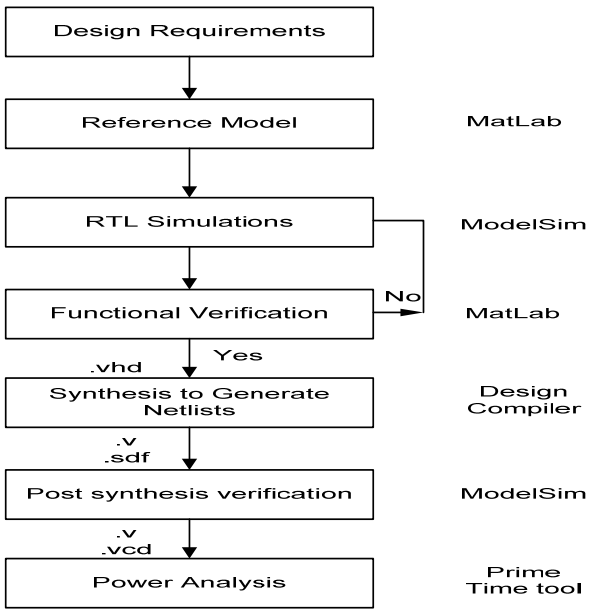

Figure 9. The Design flow

5.2 Top level architecture

In this chapter, the global architecture for complete design is explained in top down methodology. Two stage parabolic synthesis, linear interpolation, optimization techniques, and word length evaluation are explained briefly.

Fig. 10 shows the top level architecture for the design. The architecture is divided in to two parts, which is represented in dashed lines. The upper part represents a two-stage Parabolic Synthesis and lower part represents a linear interpolation. Several optimization techniques are applied to improve the hardware efficiency. The complete architecture is designed and implemented with 4 multipliers, 7 adders, 1 multiplexer, and a 2's complement module. The detailed description of every stage is explained further.

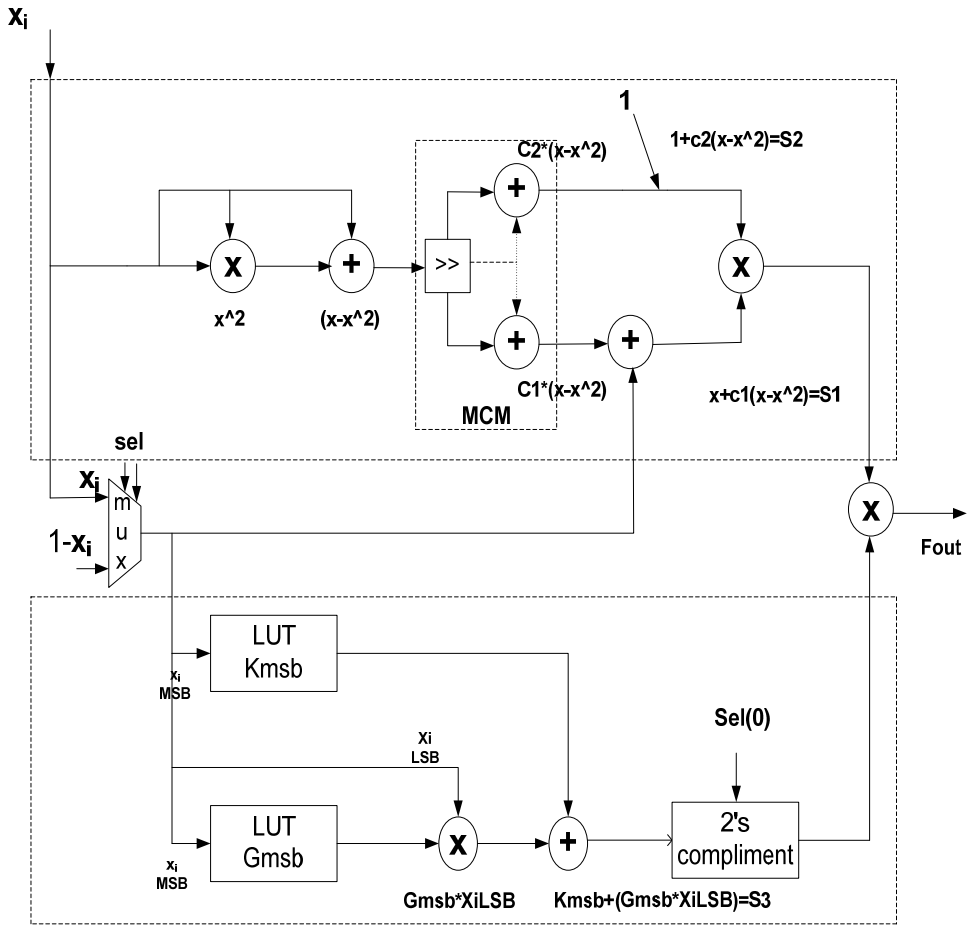

Figure 10. The Top level architecture

5.3 Parabolic synthesis architecture

5.3.1 Architecture

The implementation of sub functions in Parabolic Synthesis and optimization techniques in the architecture is discussed in this section. The implementation is divided into 3 steps namely preprocessing, processing, and post processing, which are discussed further briefly. Fig. 11 represents the architecture for parabolic synthesis.

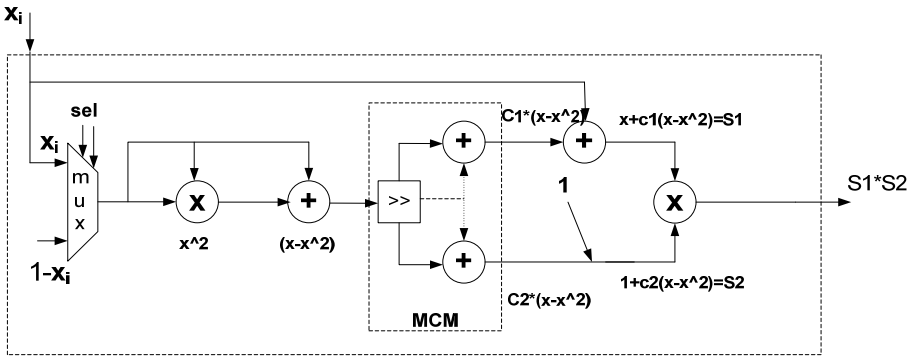

Figure 11. The Architecture of 2 Stage Parabolic Synthesis

The design consists of a multiplexer to select the inputs for a specific quadrant, and Multiple Constant Multiplier (MCM) unit to replace the multipliers with adders. This architecture results the product of two sub functions, which is multiplied to the result from the linear interpolation to get the final output.

5.3.2 Processing

Processing is the main part of the architecture, which calculates the product of two sub functions. The two parabolic functions in the methodology are easily implemented in hardware since they only need normal adders and multipliers. Applying some optimization techniques efficiently can reduce the number of cells required. Equations (2.4) and (2.6) are the two parabolic functions to be implemented in hardware and product of those two sub functions results in the output.

$$f_{par}(x) = s_1(x) * s_2(x) \tag{5.1}$$

The front end part of the design is a multiplexer, which selects the input depends on selection lines. The constants c_1 and c_2 calculated in (4.2) and (4.6) respectively are used and it's equivalent binary representation is shown below.

$$c_1 = 0.5703125_{(10)} = 10010010_{(2)} \quad (5.2)$$

$$c_2 = 0.0.400390625_{(10)} = 01100110_{(2)} \quad (5.3)$$

5.3.3 Hardware optimizations

Multipliers plays an important role in occupying the hardware area. Different optimization techniques have to be applied to avoid the usage of multipliers. In the design, two multipliers have the common multiplicand with c_1 and c_2 as multipliers. The MCM unit replaces those two multipliers with 4 adders. The MCM unit is a replacement for multipliers when they have common multiplicand, which is built with shift operations and adders, which results less hardware area.

Fig.12 shows the implementation of MCM unit with 4 adders. The MCM unit takes the value $x - x^2$ and generates the product value with both c_1 and c_2.

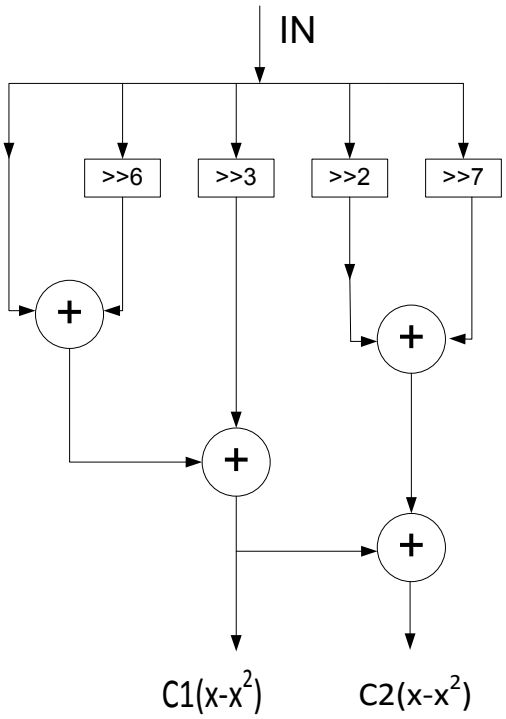

Figure 12. The Multiple Constant Multiplier (MCM) unit

The other optimization in hardware is the replacement of an adder in calculating the second sub function. In the calculation of the second sub function, the value 1 is added to c_2* (x-x^2). The adder in this can be removed and a binary '1' is padded at MSB position as presented in Fig. 11.

5.4 Linear interpolation architecture

5.4.1 Overview

In this chapter, the architecture for the third sub function with linear interpolation is explained. Fig. 13 shows the architecture for the third sub function that consists of
1 multiplier, 1 adder, and 2 Look Up Tables (LUT).

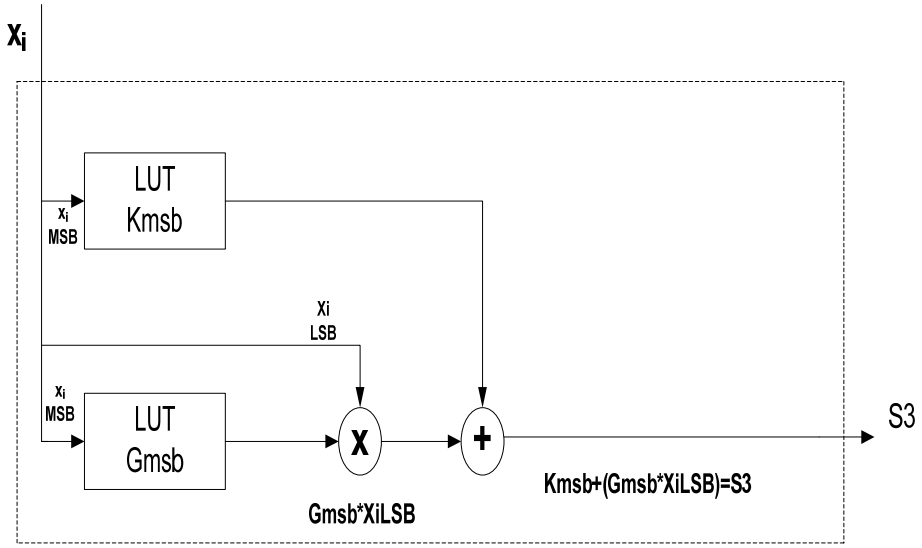

Figure 13. The architecture of the third sub function (Linear interpolation)

In the design, the values *K* and *G* are calculated from (4.3).They are stored in a LUT and the MSB of the input are used to address the position of the bits.

5.4.2 Hardware optimizations

The values of *G* from table 2 contain both positive and negative values. If they have to be used in 2's complement representation with same precision requires more number of 1's which results to more power consumption and chip area as

well. Providing some logic in the arithmetic calculations avoids this kind of representation. In this case, a dual functional adder is built, which can perform both addition and subtraction.

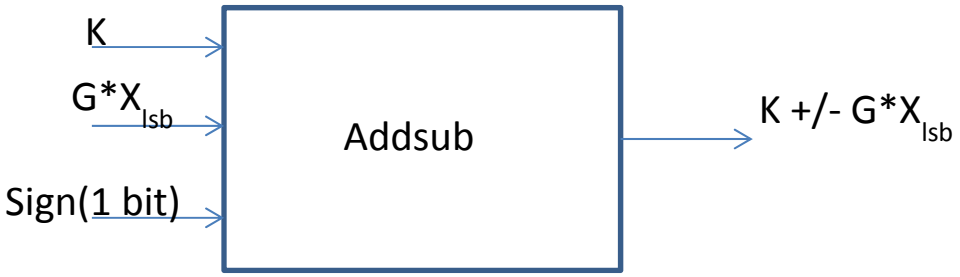

Figure 14. The Addsub Component

As shown in Fig. 14, sign bit decides the operation should be performed by the component. The first MSB in *G* is used to represent either if the value is positive or negative.

Hardware implementation of third sub function needs a multiplier and an adder. Fig.14 shows the hardware architecture of the linear interpolation method. The adder in this case is a dual functional, which means it can be switched to have either an addition or subtraction functionality. We needed this since, the product of G and X_{LSB} can be both negative and positive. Fig.14 shows the Addsub component. Because of sine function's symmetry property and by reading in reverse order, the *K* and *G* coefficients stored in LUTs can be used for all the four quadrants.

The output from linear interpolation is multiplied with the parabolic synthesis output to get the final result.

5.5 Post-processing

The value of sine function is negative in third and fourth quadrants. The post processing part transforms the output into a feasible form. The results in negative are converted into 2's complement representation. The 2's complement component is switched only in 3 and 4 quadrants and q_1 is used to enable or disable it. Fig. 15 shows block diagram of 2's complement converter.

Figure 15. Two's complement module

5.6 Word length optimization

The design is aimed for the usage of fewer bits to represent the data at different stages. The study to fix the word length at every stage is done in MATLAB and is verified, as well as for the loss of precision. Fig. 16 shows the block diagram with word length at different stages.

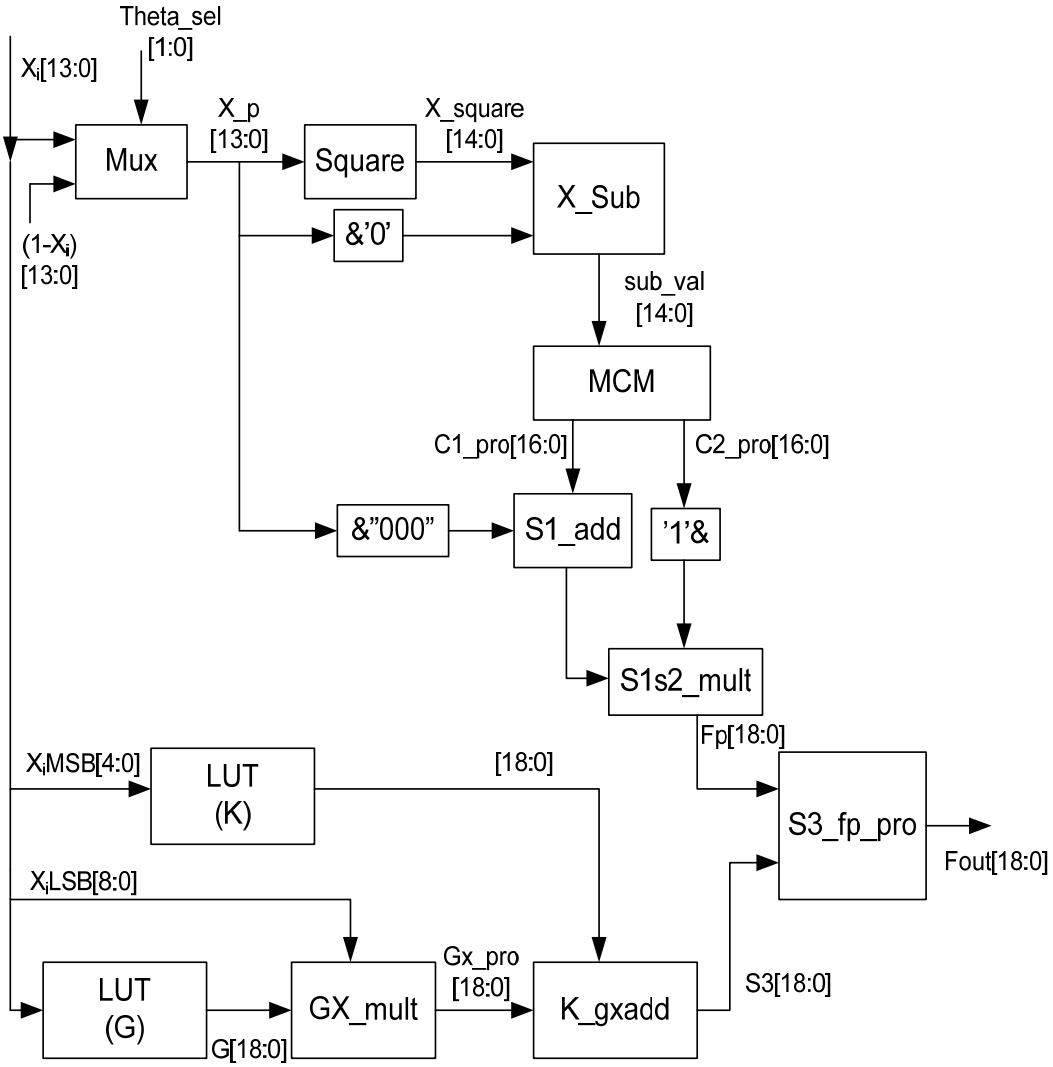

Figure 16. Block diagram with wordlengths

Table 2. Word lengths at every stage

Name	Input1(length)	Input2(length)	Original output(length)	Truncated(length)
Square	$x_i(14)$	$x_i(14)$	$x_i^2(28)$	$x_i^2(15)$
x_sub	$x_i(14)$	$x_i^2(15)$	$x_i - x_i^2(15)$	$x_i - x_i^2(15)$
MCM	$x_i - x_i^2(15)$	---------	$c_1(7) * x_i - x_i^2$	$c_1 * (x_i - x_i^2)$ (17)
			$c_2(7) * x_i - x_i^2$	$c_2 * (x_i - x_i^2)$ (17)
s_1 _ add	$x_i(14)$	$c_1 * (x_i - x_i^2)$ (17)	$s_1 = x_i + c_1 * (x_i - x_i^2)$ (18)	$s_1 = x_i + c_1 * (x_i - x_i^2)$ (18)
s_2 _ add	1	$c_2 * (x_i - x_i^2)$ (17)	$s_2 = 1 + c_2 * (x_i - x_i^2)$ (18)	$s_2 = 1 + c_2 * (x_i - x_i^2)$ (18)
s_1 s_2 mult	$s_1(18)$	$s_2(18)$	$fp = s_1 * s_2$ (36)	$fp = s_1 * s_2$ (19)
G_X_Mult	$x_{LSB}(9)$	G (19)	G * $x_{LSB}(28)$	G * $x_{LSB}(19)$
K _GX_Add	K (19)	G * x_{LSB} (19)	$S_3 = K + G * x_{LSB}$ (19)	$S_3 = K + G * x_{LSB}$ (19)
S3_fp _Mult	S_3 (19)	fp (19)	$S_3 * fp$ (38)	$S_3 * fp$ (19)

The input which is in the range 0 to 1 is represented using 14 plus 2 bits used for selecting the transformation of the input and output in case of the other quadrants. A trial and error method to fix the word lengths at every stage has been used. A function is developed in MATLAB to truncate the word length at each stage in a flexible way. A generic methodology is chosen in designing the components even in implementing in the whole design in hardware.

In Fig. 16, the word lengths at every stage are shown. The data will be truncated to a specific word length after every multiplier, since the multiplier increases the number of bits. The accuracy desired is cross verified when truncating the bits at every stage. The ***MUX*** in architecture selects any one of the 14-bit inputs using two selection lines. The selection of the inputs is needed to calculate the sine value in other quadrants. The ***squarer*** unit squares the input and the word length of the input is doubled to 28 bits. The x_i^2 is truncated to 15 bits from 28 bits. The $x_i - x_i^2$ is calculated using a subtractor. No data truncation is needed after any adders or subtractor since they does not increase word length to a bigger number , but before the operation the word lengths should be balanced in magnitude by adding zeros at LSB position, which is represented as '&0' in Fig. 16 . The ***MCM*** unit is designed to calculate the two product terms specified in table.4 according to the coefficients c_1 and c_2 , which are 7 bit length. To calculate the first sub-function s_1 ,the input x_i

is added to one of the product from ***MCM*** unit. To calculate the second sub-function s_2 , to add a value of 1,one more bit is padded to the other product term at MSB position as '1&' denoted in Fig. 16.

The multiplier ***S1S2_mult*** gives the product of two sub functions in 35 bit size, which is then truncated to 19 bits. The ***LUT*** contains *K* and *G* values, which are 19 bit word length, which are tabulated in table 3 and table 4. The LSB 9 bit in *x* is multiplied with *G* and the result is 28 bits in length, which is then truncated to 19 bits. ***K_gxadd,*** one more adder adds the *K* value to the product and results third sub function with linear interpolation approximation. The product of two sub functions and the third sub function are multiplied using a multiplier named as ***S3_fp_pro*** to produce the final result. The result is truncated to 19 bits from 38 bits. The truncation at each stage is a trial and error method keeping in mind that the desired accuracy is more than 15 bits with minimum word lengths. The word lengths of inputs and outputs at every stage and every component are tabulated in table 2.

Table 3. Values of K in Linear interpolation

Sl. no	Floating value	Binary equivalent
1	1	1000000000000000000
2	0.999275207519531	0111111111101000010
3	0.998470306396484	0111111111001101111
4	0.997886657714844	0111111110111010110
5	0.997474670410156	0111111110101101010
6	0.997238159179688	0111111110100101100
7	0.997127532958984	0111111110100001111
8	0.997154235839844	0111111110100010110
9	0.997272491455078	0111111110100110101
10	0.997489929199219	0111111110101101110
11	0.997768402099609	0111111110110110111
12	0.998119354248047	0111111111000010011
13	0.998504638671875	0111111111001111000
14	0.998935699462891	0111111111011101001
15	0.999374389648438	0111111111101011100
16	0.999839782714844	0111111111111010110
17	1.000293731689453	1000000000001001101
18	1.000747680664063	1000000000011000100
19	1.001171112060547	1000000000100110011
20	1.001579284667969	1000000000110011110
21	1.001934051513672	1000000000111111011
22	1.002258300781250	1000000001001010000
23	1.002513885498047	1000000001010010011
24	1.002716064453125	1000000001011001000
25	1.002834320068359	1000000001011100111
26	1.002880096435547	1000000001011110011
27	1.002822875976563	1000000001011100100
28	1.002677917480469	1000000001010111110
29	1.002407073974609	1000000001001110111
30	1.002025604248047	1000000001000010011
31	1.001499176025391	1000000000110001001
32	1.000835418701172	1000000000011011011

Table 4. Values of G in Linear interpolation

Sl. no	Floating value	Binary equivalent
1	-7.209777832031250e-04	(1) 101111010
2	-8.049011230468750e-04	(1) 110100110
3	-5.836486816406250e-04	(1) 100110010
4	-4.100799560546875e-04	(1)0 11010111
5	-2.384185791015625e-04	(1)00 1111101
6	-1.068115234375000e-04	(1)0000111000
7	2.288818359375000e-05	(0)0000001100
8	1.182556152343750e-04	(0)0000111110
9	2.174377441406250e-04	(0)0001110010
10	2.784729003906250e-04	(0)0010010010
11	3.490447998046875e-04	(0)0010110111
12	3.852844238281250e-04	(0)0011001010
13	4.291534423828125e-04	(0)0011100001
14	4.405975341796875e-04	(0)0011100111
15	4.615783691406250e-04	(0)0011110010
16	4.520416259765625e-04	(0)0011101101
17	4.539489746093750e-04	(0)0011101110
18	4.234313964843750e-04	(0)0011011110
19	4.062652587890625e-04	(0)0011010101
20	3.566741943359375e-04	(0)0010111011
21	3.204345703125000e-04	(0)0010101000
22	2.555847167968750e-04	(0)0010000110
23	2.021789550781250e-04	(0)0001101010
24	1.163482666015625e-04	(0)0000111101
25	4.577636718750000e-05	(0)0000011000
26	-5.531311035156250e-05	(1)0000011101
27	-1.468658447265625e-04	(1)0001001101
28	-2.689361572265625e-04	(1)0010001101
29	-3.814697265625000e-04	(1)0011001000
30	-5.264282226562500e-04	(1)0100010100
31	-6.618499755859375e-04	(1)0101011011
32	-8.354187011718750e-04	(1)0110110110

CHAPTER 6

6 Results

6.1 Test setup

The design is verified with the help of a test bench by stimulating the inputs from a text file. The output from the top module is written to a text file, which is imported to MATLAB to compare it with the reference output. Fig. 17 shows the entire setup for testing the design.

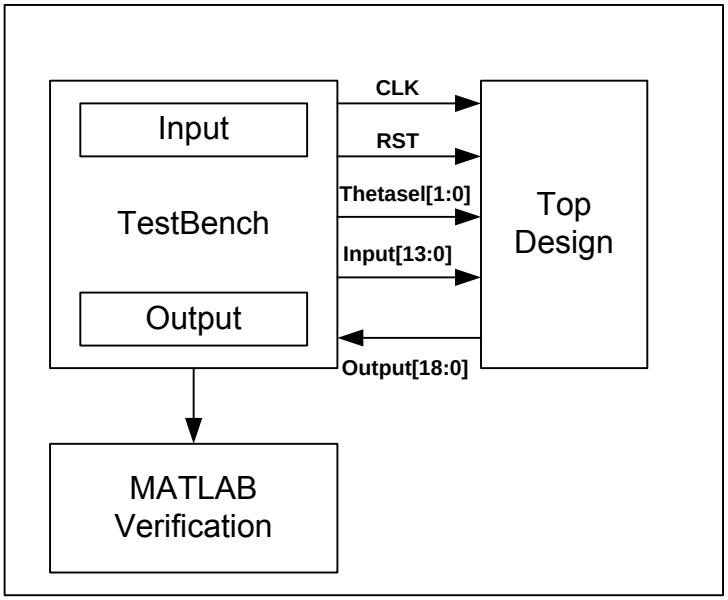

Figure 17. The Design Test setup

The absolute error for the original function and results from the hardware implementation are calculated to check the accuracy. Fig.18 shows the absolute error calculated for 1 sub function, 2 sub functions, and 3 sub functions. The accuracy is increased to more than 15 bits as the number of sub functions are increased to 3.

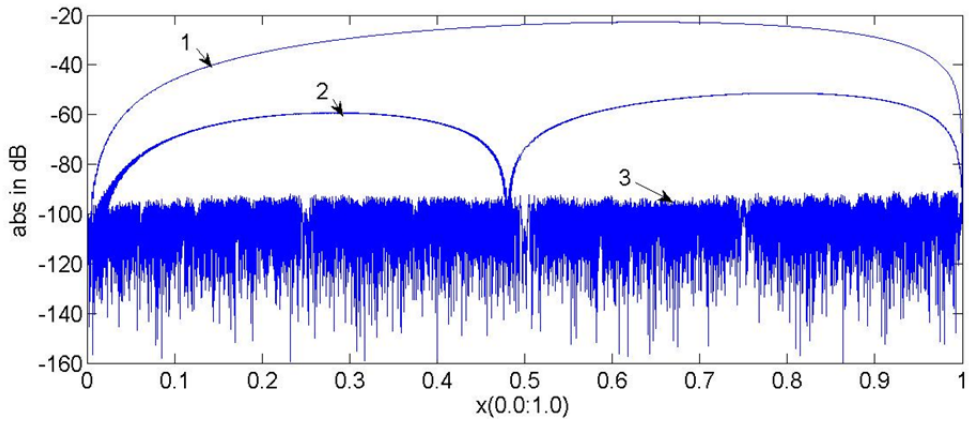

Figure 18. The absolute error of the original function compared with the result using 1, 2, and 3 sub functions

6.2 Synthesis results

Synthesis is one of the important and the basic step in the ASIC flow. Here the design tool generates gate level net lists from the Register Transfer Level (RTL). A design tool called design vision or DC-compiler by Synopsys has been used for the synthesis. The netlist is generated by using STMicroelectronics *65nm* CMOS technology. Different parameters of the design such as area, time and maximum clock frequency are analyzed. Further, this generated netlist is used for power analysis and will be explained in the chapter 6.3.

6.2.1 Design constraints

Low Power Low Threshold Voltage ($LPLV_T$) and Low Power High Threshold Voltage ($LPHV_T$) are the two different technology libraries that are used to synthesize the design. The design is synthesized using two specified libraries for different supply voltages to estimate both static and dynamic behavior. Maximum speed and minimum area are the two design constraints to the design. One is to find critical path in the design and the other to know minimum area. These constraints are applied to the design during synthesis.

6.2.1.1 Maximum Speed

The maximum speed of the design estimated by setting up the clock to 1ns, area has not set to any value, $LPHV_T$ libraries and 1.2V as supply voltage. As seen from the table 5, we have got negative slack for maximum speed optimization. A slack of 4.29*ns* is added to the clock to know the critical path time.

Table 5 .Maximum speed optimization

MEASURES	PARABOLIC SYNTHESIS	UNITS
Data Arrival Time	5.16	*ns*
Data Required Time	0.87	*ns*
Critical path slack	-4.29	*ns*
Combinational Area	19000	μm^2
Non-combinational Area	367	μm^2
Total Area	19400	μm^2

6.2.1.2 Minimum Area

Minimum area of the design is estimated by setting up the *set_maxarea* to zero during synthesis. The design is synthesized for $LPHV_T$ libraries and 1.2V as supply voltage. From table 6, we can see that the tool optimizes more on area compared to maximum speed.

Table 6 .Minimum area optimization

MEASURES	PARABOLIC SYNTHESIS	UNITS
Data Arrival Time	19.83	*ns*
Data Required Time	-17.93	*ns*
Critical path slack	1.86	*ns*
Combinational Area	11000	μm^2
Non combinational Area	343	μm^2
Total Area	11400	μm^2

Area and speed are the two major features of the design to be interested. The table 5 and 6 are the two different synthesis runs that are necessary for an area optimized design and speed optimized design. The *set_max_area* to zero applies area constraint to the design from which the area cost of the design is calculated. To analyze high speed circuit, no value should be set on area and design should run on high clock frequency. The area is increased by 70% when the design is synthesized for maximum speed and speed is increased nearly thrice.

6.2.2 Area of different modules in the design

The area occupied by different components in the design are calculated and tabulated in table 7. The values are taken from the same scenario as specified in 6.2.1.2 .We can see that most of the area in the design is consumed by the multipliers. There are four multipliers in the whole design, which have different word lengths and different areas. The ***DFFs*** are used to analyze the critical path delay in the design. The ***input MUX*** is used to select one of the inputs depending on the quadrant. There is one 14- bit multiplier (squarer in Fig. 16) and 19-bit multiplier to calculate the product of first two sub functions. These two multipliers plays a major role in area occupancy and hence the number is bigger for the ***S1, S2 Function*** module in table 7. The ***S3 Function*** also consists of one Multiplier and two LUTs to store the coefficients. A 19-bit ***multiplier*** gives the final result, which is a product of three sub functions.

Table 7 .The area of different modules in the design

Module name	Area	Units
Input DFF	147	μm^2
Output DFF	200	μm^2
Input MUX	60	μm^2
S1,S2 Function	5432	μm^2
S3 Function	2402	μm^2
Multiplier	3148	μm^2
Other cells	22	μm^2
Total Area	11400	μm^2

6.2.3 Maximum clock frequency

The speed of the design is measured in terms of maximum clock frequency, which is the sum of delays in critical path. The maximum clock frequency is calculated by the minimum time period required for the design, which we can see in table 5. The maximum clock frequency is the inverse of the total critical path time, which is calculated as below.

The critical path of the design will be affected if there is any change in the operating voltage, temperature and threshold voltage. Maximum speed of the design is calculated by setting a high frequency clock value as a constraint to the design without setting any value for area. The Synthesis scenario is specified in 6.2.1.1.

$$Maximum\ Clock\ Frequency = \frac{1}{time\ period}$$

$$= \frac{1}{5.29ns} = 189\ Mhz$$

6.3 Timing and Area analysis

6.3.1 Timing

The design is synthesized with *Synopsys design compiler* using both *65nm* $LPHV_T$ and $LPLV_T$ CMOS libraries. The timing behavior is analyzed under different scenario like operating the design at different supply voltage. Fig. 19 shows graphical description for critical path delay in both $LPHV_T$ and $LPLV_T$ cases and it is clear that the delay is almost double in case of $LPHV_T$ compared to $LPLV_T$.

Table 8. Timing values for $LPHV_T$ and $LPLV_T$

Voltage	$LPHV_T$ (ns)	$LPLV_T$ (ns)
1.1	18.4	10.0
1.15	17.0	9.5
1.25	14.3	8.78
1.3	13.2	8.4

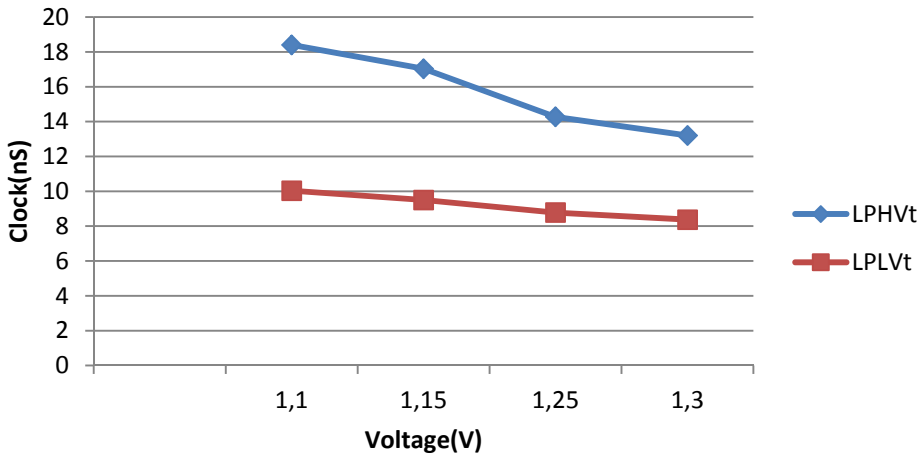

Figure 19. Timing behavior for $LPHV_T$ and $LPLV_T$

6.3.2 Area

The Parabolic Synthesis methodology architecture is designed focusing on the area occupied on hardware. The major components in the design, which occupies the area are multipliers and the optimizations in the design, which are explained chapter 5.3.3, reduce the area of the design. Fig. 20 shows the graphical description of area, which is analyzed under different supply voltages i.e. 1.1V, 1.15V, 1.25V and 1.3V for both $LPHV_T$ and $LPLV_T$ libraries.

Table 9 . Area for $LPHV_T$ and $LPLV_T$

Voltage	$LPHV_T$ (μm^2)	$LPLV_T$ (μm^2)
1.1	11338	11470
1.15	11331	11580
1.25	11327	11743
1.3	11317	11820

The values in table 9 are calculated for minimum area design by keeping set_*max_area* to 0 during the synthesis. From Fig. 20, there is a slight decrement in area of the design as the supply voltage is increased in case of $LPLV_T$,where as it is opposite in case of $LPHV_T$.

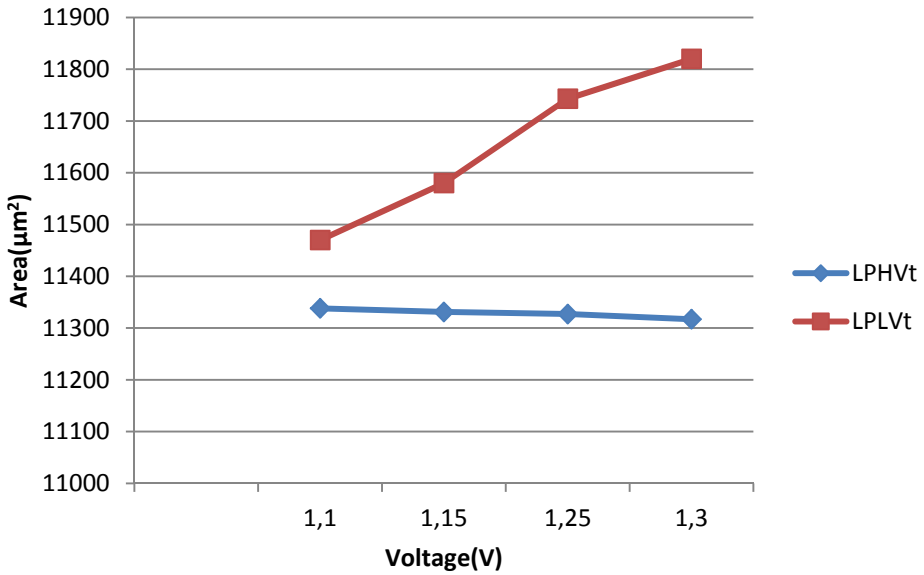

Figure 20. Area analysis for $LPHV_T$ and $LPLV_T$

6.4 Power analysis

6.4.1 Dynamic Power

The dynamic power depends on the switching activities in the design, capacitance, operating frequency and voltage as equated in (6.1).

$$P_{dyn} = \frac{1}{2}\alpha c f v^2 \quad (6.1)$$

where α = Switching activity (0-1, 1-0)
c = node capacitance
f = clock frequency
v = supply voltage

The power consumption for the design in case of both $LPHV_T$ and $LPLV_T$ is analyzed using the Primetime tool. The total power is divided into static power and dynamic power and the major contribution for dynamic power is switching activities of the design. The netlist file generated after synthesis is used in this process and the switching activities are generated as a Value Change Dump (VCD) file in the post synthesis verification process. The dynamic power consumed by the design, for different voltages in both cases is plotted in Fig. 21 and tabulated in table 10 and 11.

Table 10. Power results ($LPHV_t$)

Voltage(V)	Cell internal Power(µW)	Net switching power(µW)	Leakage power(µW)	Total Power(µW)
1.1	70.26 (46.5%)	74.11 (49.2%)	6.5 (4.3%)	150.9 (100%)
1.15	77.03 (46.6%)	81.23 (49.1%)	7.2 (4.3%)	165.5 (100%)
1.25	91.9 (46.7%)	96.4 (48.9%)	8.67 (4.4%)	197.0 (100%)
1.3	101.4 (46.9%)	105.3 (48.7%)	9.4 (4.4%)	216.2 (100%)

Table 11. Power calculation ($LPLV_t$)

Voltage(V)	Cell internal Power(µW)	Net switching power(µW)	Leakage power(µW)	Total Power(µW)
1.1	87.3 (13.9%)	84.3 (13.4%)	450 (72.7%)	626.7 (100%)
1.15	101.8 (14%)	86.29 (11.8%)	540 (74.2%)	730 (100%)
1.25	131.4 (13.4%)	104 (10.6%)	740 (76%)	983 (100%)
1.3	147.3 (13%)	113.5 (10%)	870 (77%)	1133 (100%)

The dynamic power calculated is plotted in Fig. 21 and dynamic power is more in case of $LPLV_t$ compared to $LPHV_t$.

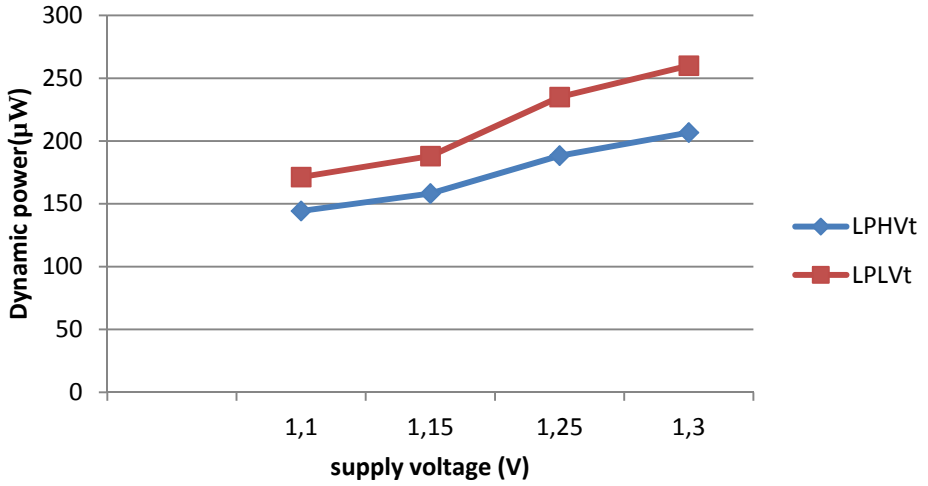

Figure 21. The dynamic power consumption ($LPHV_T$ &$LPLV_T$)

6.4.2 Static Power

The leakage power is mainly depends on transistor width, threshold voltage and zero threshold leakage current as shown in (6.2).

$$p_{static} = (I_{sub} + I_{ox}) . V_{dd} \quad (6.2)$$

The leakage power at different supply voltage for both $LPHV_T$ and $LPLV_T$ is analyzed and plotted in Fig.22 and 23. The leakage power is more in case of $LPLV_T$ compared with $LPHV_T$.

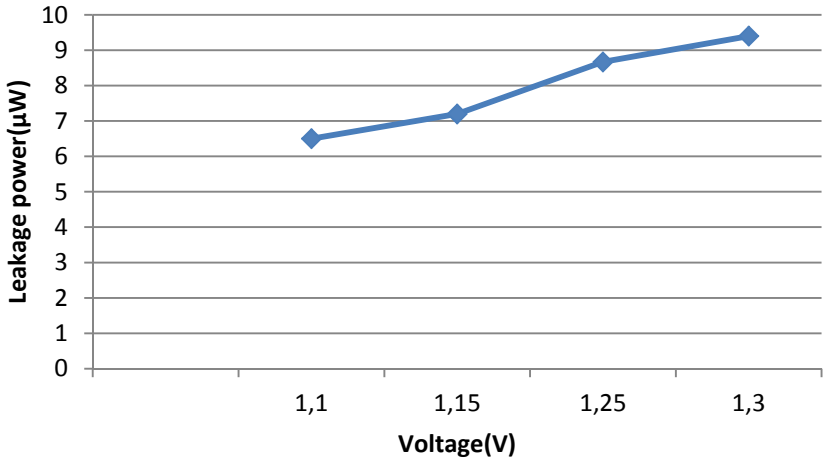

Figure 22. The Leakage power ($LPHV_T$)

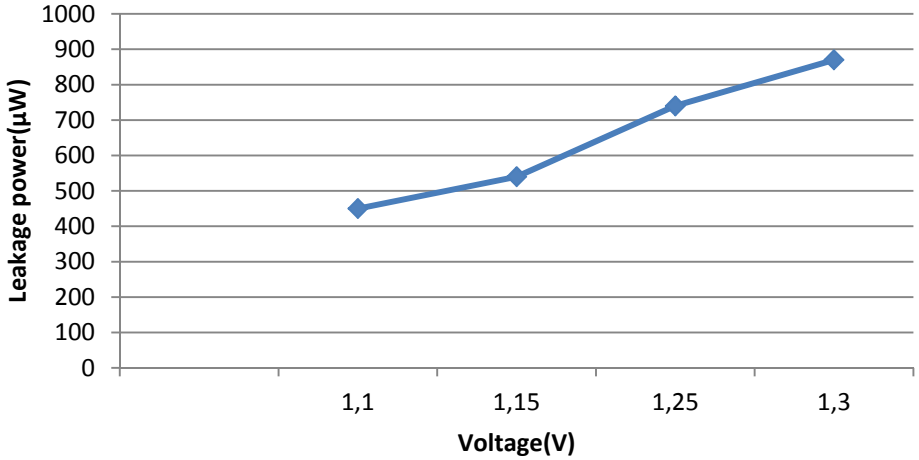

Figure 23. The Leakage power ($LPLV_T$)

The static powers in case of both technology libraries are plotted in Fig.22 and 23. The static power is very less in case of $LPHV_T$ where it is a major contribution in total power in case of $LPLV_T$. In case of $LPLV_T$, there is rapid growth in static power consumption as the supply voltage is increasing.

6.5 Energy

The power dissipation is the main objective when analyzing the efficiency of a design. The main source of power includes the switching activities, leakage current and short circuit current. The switching activities are dynamic in nature and leakage current depends on the threshold voltage. To analyze the efficiency of the design, energy consumed by the design is considered as a measure as calculated according to (6.3).

$$energy = p_{avg} \cdot time \tag{6.3}$$

The design is analyzed for energy consumption in case of both $LPHV_T$ and $LPLV_T$ libraries. The values in table 12 are measured at the maximum frequency of the design. The energy consumption is less in case of $LPLV_T$ compared to $LPHV_T$.

Table 12. Energy consumption in the design

Voltage	Energy consumption(nJ)$LPHV_T$	Energy consumption(nJ)$LPLV_T$
1.1	2.65	1.71
1.15	2.67	1.78
1.25	2.68	2.06
1.3	2.71	2.18

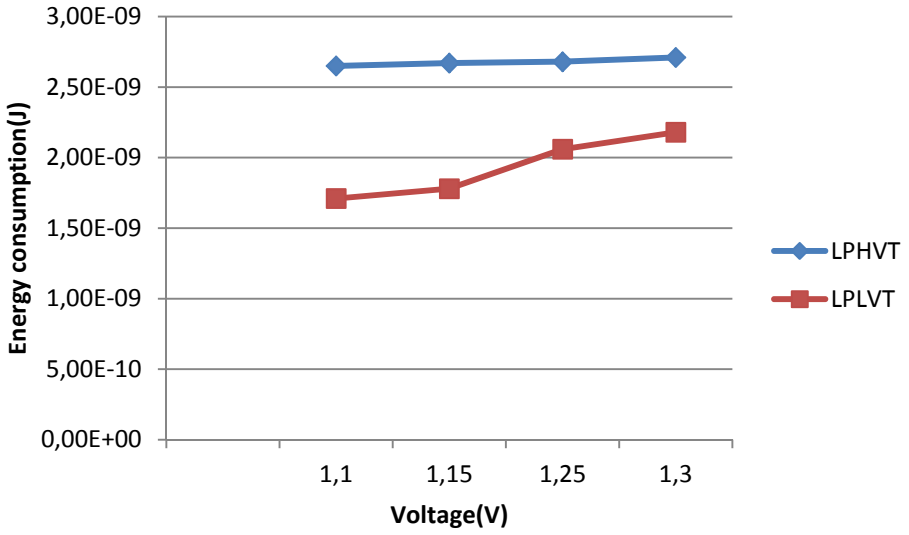

Figure 24 . Energy consumption in the design

From Fig. 24, the energy consumption is less in case LPLVT compared to LPLVT. Since the leakage power is more in LPLVT, it is not considered to be good approach to proceed to LPLVT libraries. A tradeoff is applied when considering the libraries when synthesizing the design.

6.6 Comparison

The sine function approximation using Parabolic Synthesis combined with linear interpolation results are compared with other implementations. In this chapter, the differences between the implementations and comparing the static and dynamic behavior of the designs.We considered the implementation of exponential function using Parabolic Synthesis methodology [13] and compared with the sine function approximation design. The Parabolic Synthesis methodology provides a feasibility to implement any unary function by using the same architecture with different coefficients. Peyman implemented the exponential function using Parabolic Synthesis using four sub functions in order to get more than 15 bits resolution. The same accuracy is achieved in our implementation with two sub functions and linear interpolation. The CORDIC algorithm is used to calculate the Trigonometric, and the Logarithmic functions, where the application demands simple hardware and low precision as well. The CORDIC algorithm only consists of adders, shifters and a lookup table and hence it occupies less hardware area [19].The algorithm approximately increase the precision by 1 bit per iteration, which means at least 15 iterations are needed achieve a 15 bits precision[19].

Table 13. ASIC Synthesis analysis

Methodology	Area(um^2)	maximum operating frequency(Mhz)
CORDIC	19150	11.5
Parabolic Synthesis(4 Stage)	26400	48
Parabolic Synthesis(2 stage) with Interpolation	19400	55

Table 14. Total Dynamic power analysis

Methodology	Cell internal power (µW)	Switching power (µW)	Leakage power(nW)
CORDIC	132.76 54%	114.61 46%	64.71
Parabolic Synthesis(4 stage)	162.04 51%	158.57 49%	97.78
Parabolic Synthesis(2 stage) with Interpolation	76.99 49.8%	77.57 50.1%	23.78

The area of the design with four sub functions is high since third and fourth sub function requires more computations. The CORDIC algorithm occupies less area but high latency makes it to use in low speed applications. The power consumption is less in CORDIC when compared with Parabolic Synthesis Methodology and higher when maximum frequency of operation is considered [13] [19].The Parabolic Synthesis combined with linear interpolations is designed to calculate the values for all the four quadrants using the same interpolation coefficients, which is also an added advantage with the design and causes that its area increases slightly in relation to the other implementations.

Table 13 and 14 shows the comparison of both the implementations in different aspects. The maximum operating frequency for the design is imported from table 8, the inverse of 18.4 ns. We cannot compare both the designs in numbers since the switching activities used to analyze the designs may not be same and hence we compared in sense of percentage of the total power.

CHAPTER 7

7 Conclusion

The approximation of the sine function is successfully implemented using the Parabolic Synthesis methodology combined with a linear interpolation technique. An accuracy of more than 15 bits is achieved with hardware implementation of the design. Hardware optimization techniques, which replace multipliers with an MCM unit, removal of adder result lesser area and power in the design. The design is synthesized in different scenario to estimate both dynamic and static behavior and results are compared with the CORDIC implementation [13] [16]. It can be concluded that the Parabolic Synthesis Methodology is efficient only in computational speed with almost 4.8 times faster. Parabolic Synthesis shows high degree of parallelism, which results shorter critical path in the design. Parabolic Synthesis methodology has flexibility to change the design to implement for other unary functions by modifying the set of coefficients.

CHAPTER 8

8 Future work

The linear interpolation technique combined with the parabolic synthesis methodology can be used for implementing other unary functions as well. Sine function can also be approximated by replacing the linear interpolation technique with other interpolation methods. The accuracy may be increased by approximating sine function with second-order interpolation technique.

CHAPTER 9

9 References

[1] Erik Hertz, "*Parabolic Synthesis*", Licentiate thesis in engineering

[2] P.T.P Tang, *Table-lookup algorithms for elementary functions and their error analysis,* Mathematics and Computer science division, Argonne National Laboratory.

[3] www.wikipedia.org/wiki/Polynomial_interpolation

[4] Chi-Chou Kao, *High-performance CORDIC rotation algorithm based on look-ahead techniques* International Journal of Electronics Vol. 98, No. 8, August 2011, 1075–1089

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

[6] Volder J. E. The CORDIC trigonometric computing technique. In IRE Trans. Electronic Computing, volume EC-8, pages 330 - 334, 1959.

[7] Venkatesh Jakke, Low Leakage Digital Design, a CORDIC Processor, Master of Science thesis.

[8] http://spiral.net/index.html

[9] http://en.wikipedia.org/wiki/Linear_interpolation

[10] www.wikipedia.org/wiki/Interpolation#Piecewiseconstant_interpolation

[11] http://en.wikipedia.org/wiki/Interpolation#Polynomial_interpolation

[12] Design of FFTs using CORDIC and Parabolic Synthesis for the twiddle factors by Muhammed Waqas Shafiq Nauman Hafeez

[13] 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," in the Proceedings of the *European Conference on Circuit Theory and Design (ECCTD 2011)*, August 29-31, 2011, pp. 729-732, Linkoping, Sweden

[14] www.ece.neu.edu/students/sbabaeiz/paper_poster/interpolation.htm

[15] http://www.mathsisfun.com/gradient.html

[16] Master's Thesis Report, A VLSI Implementation of Logarithmic and Exponential Functions Using a Parabolic Synthesis Methodology, which is Compared to the CORDIC Algorithm By Peyman Pouyan

[17] A. Volnei A. Pedroni, Digital Electronics and Design with VHDL.

[18] http://repositories.lib.utexas.edu/bitstream/handle/2152/1472/arbaughj 20424. Pdf.

[19] Orri Tomasson, Implementation elementary functions for a fixed point SIMD DSP Coprocessor, Examensarbete, Linkopoing.

CHAPTER10

10 List of Figures

CHAPTER 11

11 List of Acronyms

CORDIC	Coordinate Rotation Digital Computer Algorithm
ASIC	Application Specific Integrated Circuit
MCM	Multiple Constant Multiplication
$LPLV_T$	Low Power Low Threshold Voltage
$LPHV_T$	Low Power High Threshold Voltage
VHDL	Very High Speed Integrated Circuits Hardware Description Language
MSB	Most Significant Bit
VCD	Value Change Dump
CMOS	Complementary Metal Oxide Semiconductor
RTL	Register Transfer Level
SDF	Standard Delay Format