An FPGA Approach for SNR Estimation Using Phase-Only Data

A thesis submitted in partial fulfillment
of the requirements for the degree of
Masters of Science in Computer Systems Engineering

By

Pramodini Arramreddy, B.S.E.E.
Andhra University, 1997

August 1999
University of Arkansas

An FPGA Approach for SNR Estimation Using Phase-Only Data

This thesis is approved for
recommendation to the
Graduate Council

Thesis Director:

_____
Dr. Mitch Thornton

Thesis Committee:

_____
Dr. David L. Andrews

_____
Dr. Jonathan Simonson

Thesis Duplication Release

I hereby authorize the University of Arkansas Libraries to duplicate this thesis when needed for research and/or scholarship.

Agreed _____

Refused _____

Table of Contents

1. Introduction 1
    1.1. Motivation 1
    1.2. Thesis Description 2
    1.3. Organization of Thesis 3

2. SNR Estimation 4
    2.1. SNR Estimation Techniques 4
    2.2. SNR Estimation using phase-only data 5

3. CORDIC Theory 6
    3.1. Number Systems 6
        3.1.1. Word Format 7
        3.1.2. Decimal Numbers 7
        3.1.3. Angle Representation 7
    3.2. CORDIC Algorithm 11
        3.2.1 Vector Rotations 11
        3.2.2. CORDIC Equations 13
        3.2.3. Scale Factor (K) 13
    3.3. Modes of Rotation 13
        3.3.1. Coordinate Systems 14
        3.3.2. Rotation Modes 15
    3.4. Computed Functions 17
        3.4.1. Directly Computed Functions 18
        3.4.2. Indirectly Computed Functions 19
    3.5. Limitations, Accuracy and Error Analysis 20
        3.5.1. Limitation Range 20
            3.5.1.1. Correction Iteration 20
            3.5.1.2. Number System 21
        3.5.2. Accuracy 21
        3.5.3. Error Analysis 23
    3.6. CORDIC Examples 23
        3.6.1. Rotation Mode 23
        3.6.2. Vectoring Mode 24
    3.7. Review 25

4. Programmable Logic Devices 27
    4.1. Reason for FPGA Approach 27
    4.2. FPGA Background 28
    4.3. FPGA Technologies 30

4.3.1. Xilinx XC4000 Family 31
4.3.2. Lucent ORCA Series 3 Family 33
4.3.3. Actel 54SX Family 35
4.3.4. Altera FLEX 10KE Family 38
4.3.5. Quicklogic PASIC 3 Family 41
4.4. Hardware Partitioning 42
4.4.1. FPGA Partitioning Considerations 42

5. SNR Estimation using Phase-Only Data 45
5.1. Motivation 45
5.2. Digital Receiver 45
5.3. Theory and Analysis of SNR Technique 46
5.3.1. Calculating the Phase Margin 46
5.3.2. Estimating the Ambiguity Rate 49
5.3.3. Signal to Noise Ratios and Phase Error 50
5.3.4. Conclusions 56
5.4. Review 56

6. Implementation and Results 57
6.1. CORDIC Tradeoff 57
6.1.1. Bit-Serial CORDIC Design 58
6.1.2. Simulations 59
6.1.3. Logic Synthesis for Field Programmable Gate Arrays 59
6.1.4. Speed versus Area Tradeoff 60
6.2. Block Diagram 62
6.3. SNR Estimation 64

7. Conclusion 66
7.1. Thesis Results 66
7.2. Future Directions 67

Bibliography 68

Appendix A - Source Code 71

1. Introduction

1.1. Motivation

In Digital Communications systems, Signal to Noise Ratio (SNR) is an important input parameter for a variety of equalization and decoding algorithms. There has been some work published on the problem of SNR estimation. An initial effort was accomplished by Benedict and Soong in 1967 [19]. They "computed the SNR from higher-order averages of the envelope of a modulated signal" [26]. Next, an article about the "correlation properties of signal and noise in narrowband channels" [26] was published by Shah and Hinedi [20]. Later, Pauluzzi and Beaulieu [21] analyzed several SNR estimation techniques and studied the comparison between them.

The work presented here is unique since there are various SNR estimation techniques where the parameter is estimated using the amplitude of a signal [19], [22]. This thesis presents the idea of developing an SNR estimation circuit using phase-only data. This is achieved by using the CORDIC [1], [6] (COordinate Rotation DIgital Computer) algorithm which calculates the phase of a complex vector. This technique is useful in applications such as transmission over fading channels, where both current signal and current noise powers are unknown.

1.2. Thesis Description

The objective of this thesis is to develop a prototype SNR estimation circuit to show the feasibility and performance of the SNR estimation algorithm by implementing in an FPGA. This serves as an extended application for CORDIC rotator. An algorithm that uses minimum operations such as shifting and adding to compute arithmetic operations such as trigonometric, linear and hyperbolic functions is known as CORDIC algorithm. This was the result of work done by Jack Volder [1] in 1950's. Later, CORDIC was extended to cover many directly and indirectly computed functions which are discussed later in this report.

"CORDIC is a class of iterative solutions for trigonometric and other transcendental functions" [5]. All the trigonometric functions are computed using vector rotations. These vector rotations can be used for polar to rectangular and rectangular to polar transformations which in turn can be used in real-time navigation problems like radar signal processing in which the complex data is used during processing in Cartesian form and the target computer requires the output to be in polar form. In this case, a vector magnitude processor is used to convert from Cartesian space to polar form. In this work, the SNR estimator uses the CORDIC rotator to find the phase of the incoming signal which is represented in rectangular form (separate "in-phase" and "quadrature" channels, I and Q).

The reason this design is implemented in an FPGA based circuit is to study the performance of the SNR estimation circuit. Field Programmable Gate Arrays (FPGAs) have emerged as a matured technology and are changing the way electronic systems are designed and implemented. The components of this user-programmable hardware

device lie uncommitted on an already fabricated chip and can be programmed by the designer to implement any kind of digital circuit [8]. The most popular FPGA architectures use either a look-up table (LUT) or a multiplexer-configuration as a basic building block.

1.3. Organization of Thesis

This thesis is roughly divided into two parts. The first part addresses the implementation of CORDIC algorithm. The CORDIC processor is designed and implemented using *VHDL*, a hardware description language used to describe digital electronic systems. This *VHDL* design is mapped into different FPGA architectures using a software synthesis tool called *Synplify* to analyze speed and area tradeoff between architectures. The second section concerns the estimation of the SNR using Phase-Only data. The technique is described and implementation details are given. The thesis concludes by giving observations of the results and outlining areas of future investigation.

2. SNR Estimation

In analog and digital communications, signal-to-noise ratio is defined as a measure of signal strength relative to background noise. The ratio is written as S/N or SNR and is usually measured in decibels (dB). “The purpose of signal-to-noise ratio (SNR) estimation is to estimate the ratio of signal power to noise power at the receiver” [26]. A significant research has been done on the problem of SNR estimation over the past decades. This chapter discusses some of the SNR estimation techniques developed by several research groups.

2.1. SNR Estimation Techniques

SNR estimation has been the topic of many research papers, some of which are listed in this section. Benedict and Soong “computed the SNR from higher-order averages of the envelope of a modulated signal” [26] by calculating carrier strength and narrowband additive noise strength seperately from N envelope samples [19]. “This estimation is useful in the radar analysis of targets containing both specular and distributed components” [19].

“Symbol signal-to-noise ratio is an important parameter in system evaluation, and more specifically in digital communications since the performance of telemetry systems can be expressed in terms of the symbol SNR” [20]. Shah and Hinedi developed “split symbol moments estimator (SSME) which is an algorithm used to estimate symbol signal-to-noise ratio (SNR) in presence of additive white guassian

noise (AWGN)" [20]. A real-time SNR estimator can be used in monitoring algorithm which helps to detect problems and fix them as they occur.

A symbol SNR estimator was introduced and studied by Gilchriest [24] that works efficiently only for the strong signal cases. This algorithm provides an SNR estimate by using the absolute value of the matched filters. The performance of the same estimator is again studied by Layland [25] for the weak signal case.

Another SNR estimator was introduced by Edbauer in which inphase and quadrature components of a demodulator are used to generate analog signals. "These voltages are proportional to total power and to noise power, respectively, and provide a symbol SNR estimate after further processing" [20].

"It is difficult to calculate the SNR if only the noisy signal is known and neither the transmitted nor the noise are known" [26]. Therefore, an SNR estimation algorithm is developed by using second and fourth-order moments. "The SNR is estimated by observing the noisy signal if only the shapes of signal and noise pdf's are known" [26].

2.2. SNR Estimation using phase-only data

Several SNR estimation techniques are discussed in the previous section. This technique which is discussed in this thesis estimates the SNR by calculating the phase error in presence of additive white guassian noise (AWGN). This phase error is calculated using the vectoring mode of CORDIC algorithm which is explained in detail in the next chapter. This technique is discussed in detail later in this report.

3. CORDIC Theory

In 1956, J. E. Volder [1] introduced a set of algorithms to calculate trigonometric and hyperbolic functions. Later in 1959, he developed a COordinate Rotation DIgital Computer (CORDIC) for the computation of arithmetic operations and trigonometric functions. In 1971, Walther [2] extended these results to a single unified algorithm with improved functionality. Since then, CORDIC has been researched and improved by a number of groups [3] - [5] and has been used in a wide number of applications such as the Intel 8087 Math Coprocessor [23], the Hewlett-Packard HP-35 calculator and various robotics applications [6]. Most commonly, the CORDIC algorithm has been used in DSP applications such as Radar Signal Processors.

CORDIC computes elementary trigonometric functions, multiplication and division using only simple addition and binary shifting. It provides an efficient way of handling calculations that are most easily described in terms of rotations or angles. This has a great influence on the hardware organization, especially in terms of circuit complexity.

3.1. Number Systems

Before going into the details of CORDIC algorithm, it is essential to understand the number system that it uses. CORDIC uses a binary number system to

represent decimal values and the angle (in degrees) of rotation. This section describes the representation of binary number system.

3.1.1. Word Format

In the following, it is assumed that all words are represented in a fixed point form. The input and output values are composed of 17 bits with 13 fractional bits, 3 integer bits and 1 sign bit (1 for -ve values and 0 for +ve values). All negative values are represented in 2's compliment form.

3.1.2. Decimal Numbers

CORDIC calculates in terms of decimal numbers, but uses binary numbers to represent them. This is possible through the use of fixed-point numbers. In this system, the left most bit is a sign bit, 1 for negative number and 0 for positive number. Each digit is half the value of the bit to its left. Figure 3.1(a) illustrates the fixed-point representation of a decimal number of word length 17. The number after the decimal point designates the fractional part. The number before the decimal point represents the integer part. In the case of 1.17, the 17 means that there are 17 binary digits in the number including one sign bit and one of them is the integer number. A 1.17 number can range between [-1 and 1]. Figure 3.1(b) illustrates 4.17 numbers. The range of 4.17 numbers is [-8 and 8].

3.1.3. Angle Representation

The same process as described above is used for Binary Angle Measurement (BAM). Usually, 1.9 numbers are used to represent angles in CORDIC. Figure 3.2(a) illustrates how these angles are represented. In a 1.9 binary angle, bit '1' represents

positive angle and ‘0’ represents negative angle. The most significant digit is 90^0 (‘1‘ for $+90^0$ and ‘0’ for -90^0). Each digit afterward represents an angle (positive or negative) which is measured based on arctangents.

1 . 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

1 —	-1	10 —	.001953125
2 —	.5	11 —	.0009765625
3 —	.25	12 —	.00048828125
4 —	.125	13 —	.000244140625
5 —	.0625	14 —	.0001220703125
6 —	.03125	15 —	.00006103515625
7 —	.015625	16 —	.000030517578125
8 —	.0078125	17 —	1.52587890625e-5
9 —	.00390625		

Ex : 0.0001101101010001 — .0625 + .03125 + .0078125 + .00390625
+ .00097656 + .000244141 + .0000015258
= 0.106690979003

Ex : 1. 0001101101010001 — - .0625 + .03125 + .0078125 + .00390625
+ .00097656 + .000244141 + .0000015258
= - 0.106690979003

3.1(a)

Since CORDIC deals with angular rotations, each time a new value is computed, a different angle is used. This poses a problem because it is not feasible to modify the processor each time a different angle is to be used. This problem can be solved by using a sequence of directions for the elementary rotations [6]. This sequence can be represented by a decision vector and these decision vectors are in turn used to represent the angle based on binary arctangents. In Figure 3.2(b), an angle series of 45^0, 26.56^0, 14.04^0, 7.125^0 and so on is being used to rotate a vector in

1 2 3 4 . 5 6 7 8 9 10 11 12 13 14 15 16 17

1 —	-8	10 —	.015625
2 —	4	11 —	.0078125
3 —	2	12 —	.00390625
4 —	1	13 —	.001953125
5 —	.5	14 —	.0009765625
6 —	.25	15 —	.00048828125
7 —	.125	16 —	.000244140625
8 —	.0625	17 —	.0001220703125
9 —	.03125		

Ex : 0001.0010010010001 — 1 + .125 + .015625 + .001953125 + .0001220703125 = 1.142700195312

Ex : 1001.0010010010001 — -1 + .125 + .015625 + .001953125 + .0001220703125 = -1.142700195312

3.1(b)

Figure 3.1 Fixed Point Representation of Decimal Numbers

1 . 2 3 4 5 6 7 8 9

1	—	90^0
2	—	45^0
3	—	26.56^0
4	—	14.04^0
5	—	7.125^0
6	—	3.576^0
7	—	1.789^0
8	—	0.895^0
9	—	0.4476^0

Ex : X.11010001 — 45 + 26.56 - 14.04 + 7.125 - 3.576 - 1.789 - .895 + .4476 = 58.8306^0

Ex : X.00110101 — -45 - 26.56 + 14.04 + 7.125 - 3.576 + 1.789 - .895 + .4476 = -52.6294^0

Ex : 1.11010001 — 90 + 45 + 26.56 - 14.04 + 7.125 - 3.576 - 1.789 - .895 + .4476 = 148.8306^0

Ex : 0.00110101 — -90 - 45 - 26.56 + 14.04 + 7.125 - 3.576 + 1.789 - .895 + .4476 = 142.6294^0

Figure 3.2(a) Binary Angle Measurement (BAM)

the first quadrant to the x-axis. First, the vector is rotated towards the x-axis by 45^0. The next rotation moves 26.56^0 in the other direction. The third rotation moves 14.04^0 again in the other direction and the rotation moves 7.125^0 closer. This process will continue using more and more smaller angles for more precise over all rotation. Each cycle is called an iteration. As seen from the figure, the last rotation is almost aligned to the x-axis.

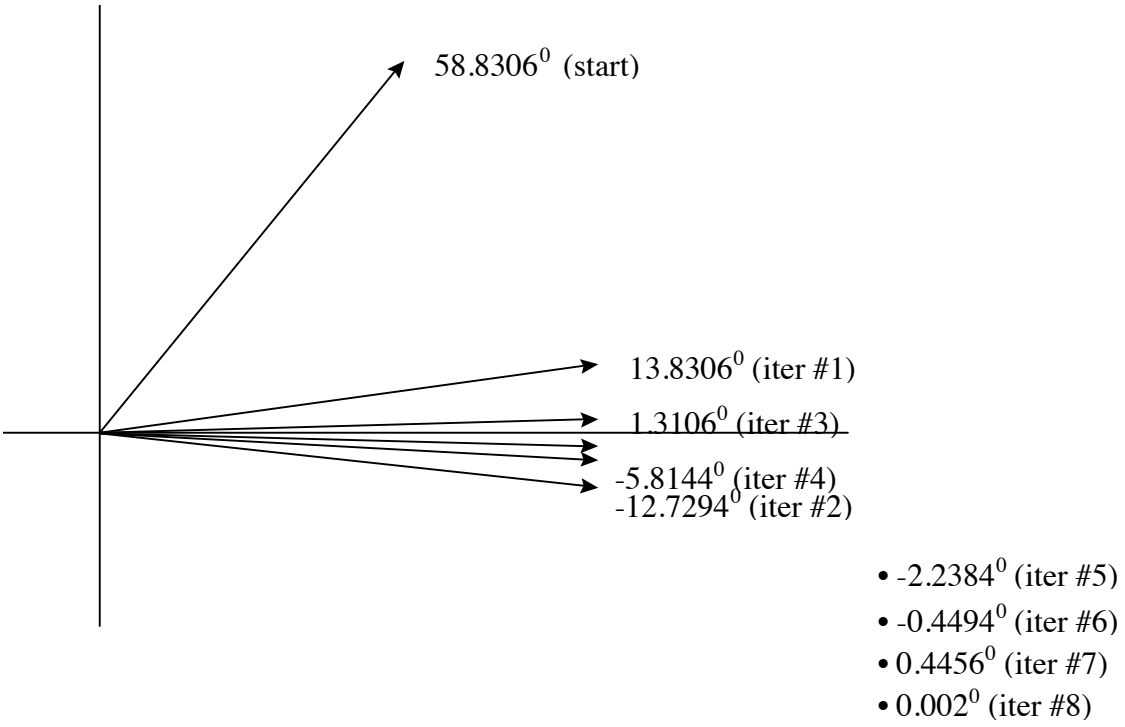

Figure 3.2(b) Angle Sequences

This method allows for the addition or subtraction of a particular angle to be made at run time. This means that only the sequence of angles has to be

predetermined. The value of the previous angle total can be used to determine whether the current angle is added or subtracted. As a result, the total number of angles that can be represented by a particular sequence of angles is equal to the number of combinations of adding and/or subtracting the sequence.

3.2. CORDIC Algorithm

This section derives CORDIC equations from the general rotation transform and also explains the scale factor.

3.2.1. Vector Rotations

Now that it has been established that different angles can be represented, the next topic to is how the new vector is to be computed. The CORDIC Algorithm is derived from the general rotation transform [6].

$$x' = x\cos\theta - y\sin\theta$$

$$y' = y\cos\theta + x\sin\theta$$

which rotates a vector [x, y] in cartesian plane by an angle θ resulting in the new vector [x', y']. The above equations can be rewritten as:

$$x' = \cos\theta\,[\,x - y\tan\theta\,]$$

$$y' = \cos\theta\,[\,y + x\tan\theta\,]$$

"If the rotation angles are restricted so that $\tan\theta = \pm 2^{-I}$ (i.e., we use BAM), then the multiplication by the tangent term is reduced to a simple shift operation" [6]. The $\cos\theta$ term is considered to be a constant, if the decision at each iteration, i, depends on the direction of vector rotation. The above equations can be now reduced to:

$$x_{i+1} = K_i[\ x_i - y_i\ .d_i\ .2^{-i}\ ]$$

$$y_{i+1} = K_i[\ y_i + x_i\ .d_i\ .2^{-i}\ ]$$

where

$$K_i = \cos\theta = \cos\ (\ \tan^{-1}2^{-i}\ ) = 1/\sqrt{1 + 2^{-2i}}$$

$$d_i = \pm 1$$

The above iterative equations results in a simple "shift-add algorithm for vector rotation" [6] by neglecting the scale constant. An angle accumulator that uses an additional adder-subtractor that accumulates the elementary rotation angles at each iteration can be used. The elementary angles can be expressed in any angular units. This results in the third difference equation for the CORDIC algorithm.

$$z_{i+1} = z_i - d_i\ .\ \tan^{-1}(\ 2^{-i}\ )$$

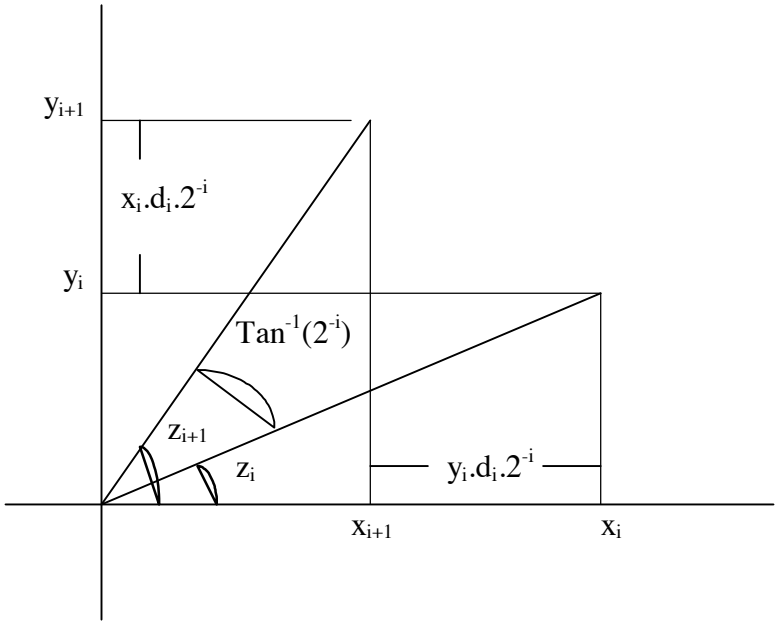

Figure 3.3 Basic Vector Rotation

3.2.2. CORDIC Equations

Now that the rotation technique has been established, it is necessary to tie the three difference equations together. The final CORDIC equations used for implementation of several trigonometric functions are given by :

$$x_{i+1} = x_i - y_i d_i 2^{-i}$$

$$y_{i+1} = y_i + x_i d_i 2^{-i}$$

$$Z_{i+1} = z_i - d_i \tan^{-1}(2^{-i})$$

3.2.3. Scale Factor (K_i)

The scale factor K_i is given by

$$K_i = 1/\sqrt{1 + 2^{-2i}}$$

"The product of K_i's can be treated as a part of system processing gain. As the number of iterations approaches infinity, the gain approaches 0.6073" [6]. Therefore

$$A_n = \lim_{i \to \infty} K_i = 1.647$$

The system gain is calculated from the expression

$$A_n = \prod_n \sqrt{1 + 2^{-2i}}$$

which depends on the number of iterations.

3.3. Modes of Rotation

There are other factors that play a part in this process. The main factors are the choice of the coordinate system and the modes of rotation. The coordinate system designates the equation that is used to determine the angle for a particular rotation. The sequence of angles are chosen correctly so that it sets the rotation of a vector in

the selected coordinate system. The selected mode of rotation helps to determine whether every angle is added to or subtracted from the total.

3.3.1. Coordinate Systems

The CORDIC algorithm is capable of rotating in three coordinate systems: Circular, Linear, and Hyperbolic. The vector rotation depends on the type of coordinate system which in turn changes the results of the overall rotation. Figure 3.4 illustrates the three different coordinate systems.

The Circular coordinate system, Figure 3.4(a), means that the top edge of the original vector lies on a circuar arc. As the vector is rotated, the top edge of the vector moves on the circular arc. Similarly, the Linear system, Figure 3.4(b) describes a vector whose top edge lies on a line. The top edge of the vector remains on the line as it is rotated. In the case of Hyperbolic coordinate system, Figure 3.4(c), the top edge of the rotated vector always lies on a hyperbolic curve.

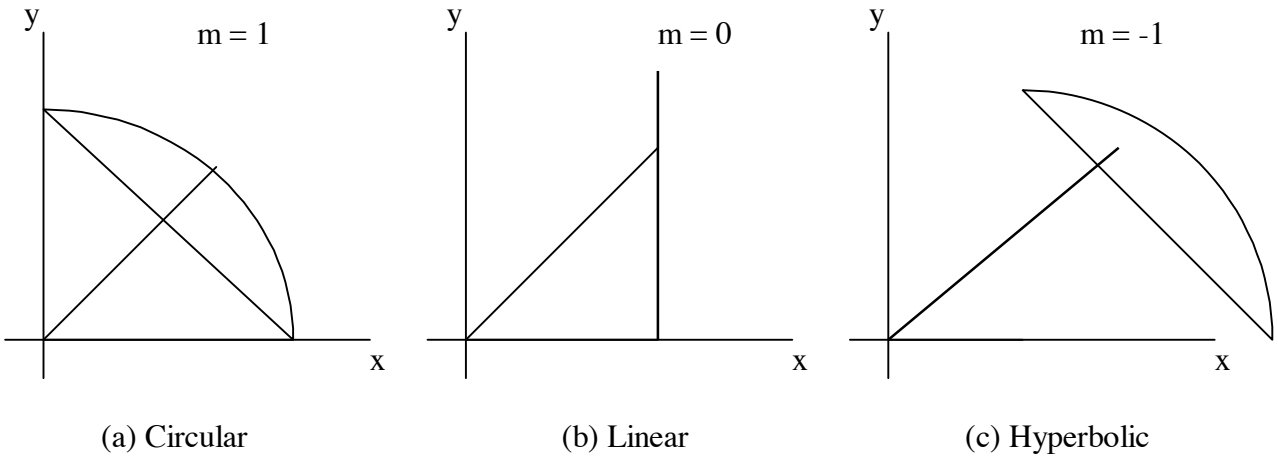

Figure 3.4 Coordinate Systems

As shown in Figure 3.4, each coordinate system will give different final vector for an initial vector rotated by the same angle. For example, if the vector located at 45^0 in every case is rotated to the x-axis, then for each coordinate system, a very different vector is calculated.

3.3.2. Rotation Modes

CORDIC calculates three values during a rotation: X, Y and Z. The X value is the x component of the vector. The Y value is the y component of the vector and the Z value is the accumulation of the sequence of angles. Thus, this algorithm operates in two types of rotation modes, rotation and vectoring.

In the rotation mode, the coordinate components of a vector (X and Y) and the angle of rotation, Z are given and the coordinate components of the original vector are calculated. The X register is always loaded with a unit vector and the Y register with the zero vector, which means that the magnitude of the given vector is always normalized with the given desired angle.

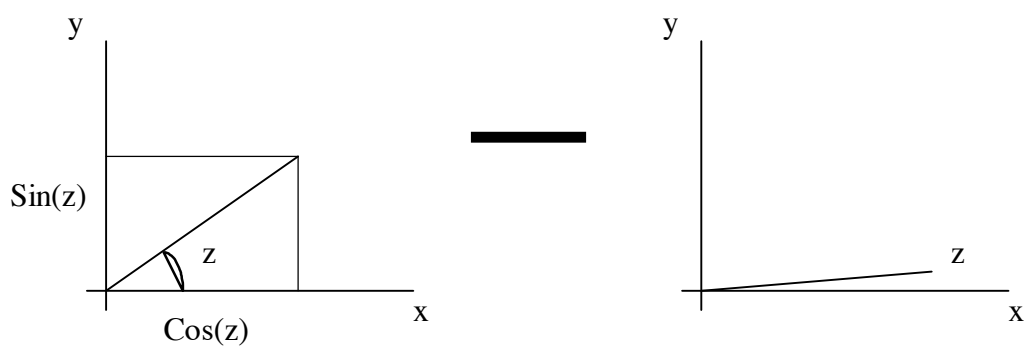

Figure 3.5 Rotation Mode

In the Figure 3.5, the X and Y components are given with normalized values with desired rotation angle, Z. At the end of the iterations, the values in X, Y ends up with the new values and a zero in Z. The value of d_i depends on the desired angle at each iteration. This is sometimes referred to as "forcing to zero".

The CORDIC equations for Rotation Mode are :

$$x_{i+1} = x_i - y_i \,.d_i \,.2^{-i}$$

$$y_{i+1} = y_i + x_i \,.d_i \,.2^{-i}$$

$$Z_{i+1} = z_i - d_i \,.\, \tan^{-1}(\, 2^{-i}\,)$$

where

$$d_i = -1 \text{ if } z_i < 0, +1 \text{ otherwise}$$

which provides the following result :

$$x_n = A_n[x_0 \cos z_0 - y_0 \sin z_0]$$

$$y_n = A_n[y_0 \cos z_0 + x_0 \sin z_0]$$

$$z_n = 0$$

In the vectoring mode, the coordinate components of a vector are given and the magnitude and angular argument of the original vector are calculated. Figure 3.6 shows the basic idea of vectoring mode. This mode starts with the component values of the initial vector in X and Y just as in the previous case. The difference is that the Z register starts out with a value of zero. This time instead of "forcing to zero" the idea is to "force Y to zero". The vectoring function works by minimizing the Y component of the residual vector at each rotation. The value of d_i is decided based on the value of Y vector after each iteration. As a result, at the end of the iterations, the Z register

contains the value of the angle by which the original vector was rotated and Y is equal to zero.

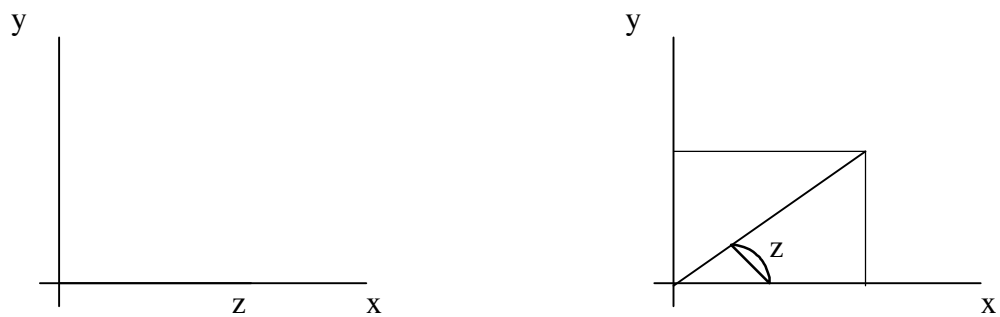

Figure 3.6 Vectoring Mode

The CORDIC equations for Vectoring Mode are :

$$x_{i+1} = x_i - y_i \,.d_i \,.2^{-i}$$

$$y_{i+1} = y_i + x_i \,.d_i \,.2^{-i}$$

$$Z_{i+1} = z_i - d_i \,.\, \tan^{-1}(\, 2^{-i}\,)$$

where

$$d_i = +1 \text{ if } y_i < 0, -1 \text{ otherwise}$$

which provides the following result :

$$x_n = A_n \sqrt{x_0^2 + y_0^2}$$

$$y_n = 0$$

$$z_n = z_0 + \tan^{-1}(y_0/x_0)$$

3.4. Computed Functions

As stated earlier, the CORDIC algorithm is capable of calculating many different trigonometric functions. Some are calculated directly, while others are computed indirectly by using directly computed functions.

3.4.1. Directly Computed Functions

Figure 3.7 illustrates the functions that can be directly computed by a CORDIC processor. By choosing a particular coordinate system and rotation mode, the different results can be selected. These functions can be further refined by selectively loading the registers. For example, if the desired function is $2*\cos(45^0)$, then the "2" is loaded into the X register, the "45" is loaded into the Z register, and "0" is loaded in the Y register. This cancels out the Y*sin(Z) part of the equation. When the process is finished, then the result can be found in X register.

One of the strengths of this system is the fact that each of the directly calculated functions requires the same amount of time to calculate. This can be very useful in pipelined applications or in highly parallel applications. These two architectures depends on a rigidly sequenced flow of data.

Many other functions can be calculated with a CORDIC processor than are shown in Figure 3.4., which means that other functions can be computed indirectly. As a result, any function that can be represented in terms of the directly computed functions can also be calculated with this processor. Some of the examples are shown below : [2]

$$\tan(z) = \sin(z) / \cos(z)$$

$$\tanh(z) = \sinh(z) / \cosh(z)$$

$$\exp(z) = \sinh(z) + \cosh(z$$

$$\ln w = 2\tanh^{-1}(y/x) \text{ where } x = w+1 \text{ and } y = w-1$$

$$\square w = (x^2 - y^2)^{1/2} \text{ where } x = w + ¼ \text{ and } y = w - ¼$$

3.4.2. Indirectly Computed Functions

Figure 3.7 Directly Computed Functions

These are certainly not the only possibilities. These functions may be computed by calculating part of the answer during the first pass through the processor and then by using these results to calculate the next part of the function. This process is continued until the desired function is computed. Very complicated algorithms can be completely implemented using CORDIC by connecting the output of one processor to the input of the other.

3.5. Limitations, Accuracy and Error Analysis

The functionality of a processor that is based upon this concept will suffer from some limitations. The major factors that contribute to this be discussed here. Each of these factors will directly affect the size, speed and complexity of the processor.

3.5.1. Limitation Range

The range of limitations means that the processor is only capable of operating within confined regions, namely quadrants I and IV. In this case, the processor's limitations are set by correction iteration and the chosen number system. Correction iteration allows to operate in all the four quadrants by additional rotation and the normalized data is used in this application to improve the precision.

3.5.1.1. Correction Iteration

"The CORDIC rotation and vectoring algorithm, as mentioned earlier, are limited to rotation angles between $-\pi/2$ and $\pi/2$. This limitation is due to the use of 2^0 for the tangent in the first iteration" [6]. An additional rotation is required to improve the rotation angles larger than $\pi/2$. This additional rotation is described by Volder[1] with an initial rotation of $\pi\pi/2$. This gives the correction iteration [6] :

$$x' = -dy$$

$$y' = dx$$

$$z' = z + d\pi/2$$

where d = +1 if y < 0, -1 otherwise

This reduction form assumes a modulo 2π representation of the input angle.

3.5.1.2. Number System

The chosen number system can be a form of limitation for a CORDIC processor. Figure 3.1 illustrates how a 17-bit binary number can represent two different ranges of values. A 1.17 number represents numbers between -1 and 1 while a 4.17 number represents numbers between -8 and 8. The tradeoff in this situation is precision. A 4.17 number represents a wider range of values but has 3 (numbers from 1-4 should be symbols) less bits of precision as a result. To settle this tradeoff, the data to be run through the processor should be examined and a number system that is application to the application should be chosen. Many previous applications have required the use of normalized data.[7,8]

3.5.2. Accuracy

There are two major factors that can have an affect upon the accuracy of a CORDIC processor. The number of rotations is another way of specifying the number of angles in the rotation sequence. In most cases, the number of rotations should be about the same as the number of bits. This is because the number of rotations can be considered to be linked indirectly to the number of shifts performed during a rotation. A large number of rotations will help to increase the precision of the overall rotation

The number of rotations should be established such that the last rotation has as many shifts as there are significant bits in the data. However, it has been observed that the precision is almost constant after eight rotations. Therefore, this implementation considers eight number of rotations eventhough it uses 16-bit data, which in turn reduces the complexity of the overall circuit.

As a preceding component is shifted and added to another number, there is a certain amount of error produced. Consider a 16 bit number that is shifted by three places to the right. The lower three bits are truncated from the number in order to keep the correct size of the number. When this new value is added to another number, a rounding error can occur due to that truncation. The error is compounded, because the error from the previous rotation is passed to the next which adds in its own error. However, this error "can be rendered harmless by using $L + \log_2 L$ bits for the storage of intermediate results" [2 p. 383], where L is the number of significant bits in the number. For example, if a 16 bit number is given as an input to the system, then the number of bits used in the intermediate steps should be $16 + \log_2 16 = 20$ bits. When the answer is computed, the lower four bits can be truncated from the answer.

This process is not necessary if it is not critical that the answers be exact to the last digit. The difference between 16 bits and 20 bits in a large system could mean a substantial increase in the size of the processor in terms of required logic. A choice must be made between accuracy and system size.

3.5.3. Error Analysis

Not only is truncation error due to the truncation of the LSBs as stated in the above section responsible, but there is angular error caused by the rotation angle. When the rotation angle lies exactly on the x-axis, there is no angular error. However, in the practical sense, the rotation angle will not be exactly zero. Since this implementation performs eight CORDIC iterations, the maximum error observed in phase angle of the resultant value is $\pm 0.408^{0}$.

3.6. CORDIC Examples

For better understanding of CORDIC algorithm, this section will present examples of rotation mode and vectoring mode calculations performed in decimal number system.

3.6.1. Rotation Mode

Table 3.6.1 contains an example of rotation in circular coordinate system using rotation mode[6]. Notice that the Z register is forced to zero at the end of the iterations. At each stage, the decision to add or subtract is based on the sign of the residual rotation angle forcing the Z register closer to zero. The MSB of the Z vector will not participate in the iteration process, since it is used to represent the angles outside the range $[-\pi/2, +\pi/2]$.

The CORDIC equations associated with the rotation mode are used to compute X, Y, and Z components. In the first stage, the X and Y components are determined based on the sign of the desired angle. The values added/subtracted to X

Iteration, i	2^{-i}	X register	Y register	Z register
		1.0000000	0.0000000	60^0
0	1.0000000	1.0000000	1.0000000	15^0
1	0.5000000	0.5000000	1.5000000	-11.565^0
2	0.2500000	0.8750000	1.3750000	2.4712^0
3	0.1250000	0.7031250	1.4843750	-4.65382^0
4	0.0625000	0.7958988	1.4404296	-1.077485^0
5	0.0312500	0.8409114	1.4155572	0.712425^0
6	0.0156250	0.8187933	1.4286964	-0.182748^0
7	0.0078125	0.8299549	1.4229957	0.264866^0

Table 3.6.1 Rotation Mode Example

and Y are the shifted values of the previous components. In this stage, the shift values have been shifted by a value of 0. In the second stage, the components are shifted by a value of 1. This process continues with the sequence of 2, 3, 4, 5, 6, and 7.

After the last stage, correction iteration is used if the angle lies outside the range [-□/2, +□/2]. It is a perfect rotation because it is rotated by 90^0. If rotating $+90^0$, then X = -Y and Y = X. If rotating by -90^0, then X = Y and Y = -X. The resultant X and Y components contain the components of the newly rotated vector. The resultant components are multiplied by the scale factor to produce desired X and Y vectors.

3.6.2. Vectoring Mode

Table 3.6.2 contains an example of rotation in circular coordinate system using vectoring mode[6]. Vectoring mode operates in the similar manner that the rotation

mode does. However, the difference is that instead of forcing the Z register to zero, the Y

Iteration, i	2^{-i}	X register	Y register	Z register
		0.5000000	0.8660000	0^0
0	1.0000000	1.3660000	0.3660000	45^0
1	0.5000000	1.5490000	-0.3170000	71.565^0
2	0.2500000	1.6282500	0.0702500	57.5287^0
3	0.1250000	1.6370312	-0.1332813	64.65372^0
4	0.0625000	1.6453612	-0.0309668	61.07738^0
5	0.0312500	1.6463289	0.0204507	59.28746^0
6	0.0156250	1.6466485	-0.0052781	60.18264^0
7	0.0078125	1.6466897	0.0075862	59.73503^0

Table 3.6.2 Vectoring Mode Example

register is forced to zero. In this case, the first rotation is used for correction iteration. Each calculation after that uses the traditional shift sequence to calculate the shift angle and the shifted value that is added/subtracted to X and Y.

3.7. Review

The CORDIC algorithm iteratively calculates the rotation of vectors. It uses a precomputed sequence of angles that is added to or subtracted from the total to represent the angle to be rotated. The chosen coordinate system determines the equations used to calculate this sequence of angles.

The algorithm is capable of computing many functions, some directly and some indirectly. The choice of coordinate system and the choice of rotation mode determines which functions are computed. All of this is accomplished with very simple hardware and with minimum operations such as shifts and adds.

4. Programmable Logic Devices

With the growing complexity of the digital circuits, designing a digital system manually is cumbersome and slow and becomes critical to minimize the design manufacturing test cycle time. Recently, user programmable gate arrays called Field-Programmable Gate Arrays (FPGAs) have emerged as an efficient technology to reduce the hardware and production cost and time. This chapter discusses the reason to implement in an FPGA and also describes different FPGA technologies.

4.1. Reason for FPGA Approach

Software programmable components such as microprocessors and digital signal processors have been used extensively in digital electronic systems, since they provide for quick design revisions. However, the inherent performance limitations of software-programmable systems make them inadequate for high-performance designs. As a result, designers turned to a mask-programmable hardware solution, namely gate arrays. Recently, user-programmable gate arrays, called field-programmable gate arrays (FPGAs), have emerged and are changing the way electronic systems are designed and implemented [9].

FPGAs have become one of the fastest growing devices in the field of digital electronics [13]. There are many reasons for their growing popularity. They are reaching speeds and densities comparable with Application Specific Integrated Circuits

(ASIC) without the associated development time and costs. They are reusable in much the same way that discrete logic is reusable, but they are faster and less costly.

Simple inexpensive FPGAs are now widely used to replace discrete logic circuits. More advanced chips are being used to verify designs while the equivalent ASIC is being developed and produced. This procedure can be very cost effective because finding design flaws before producing the ASIC can save months of delay. In some cases, the cost of FPGAs is comparable enough with ASICs such that the FPGAs are used instead.

4.2. FPGA Background

In the electronics industry, production time has become very essential since it is important to reach the market with a new product in the shortest possible time. Therefore, FPGAs have emerged as the ultimate solution to these high-risk problems. An FPGA solution provides for less manufacturing time and low cost prototypes. A field programmable device is a device in which the logic structure can be directly configured by the end user without the use of an IC fabrication facility. FPGAs have become one of the fastest growing devices in the field of digital electronics.

Before to the evolution of FPGAs, designers produced a technique called Mask Programmable Gate Arrays (MPGAs) for high performance designs. MPGAs consist of rows of transistors that can be interconnected to implement a desired logic circuit. The main advantage of MPGAs is that they provide a general structure that allows for the implementation of larger circuits because of their user specified

Figure 4.1 Conceptual Diagram of FPGA [14]

interconnection structure. However, at the same time, they require high manufacturing time and incur high initial costs. A Field Programmable Gate Array (FPGA) combines the programmability of a PLD and the scalable interconnection structure of an MPGA.

FPGAs were first introduced in 1985 by the Xilinx company. Since then, many different FPGAs have been developed by a number of companies. FPGAs are user programmable or field programmable hardware devices. They are prefabricated as arrays of identical programmable logic blocks with routing resources and are configured by the

user into the desired circuit functionality. They are now widely used to replace discrete logic circuits with higher logic density. Figure 4.1 shows the conceptual diagram of a typical FPGA. As shown in figure, it consists of a two dimensional array of logic blocks

connected by interconnection resources. These interconnection resources contains programmable switches that help to connect the logic blocks to wire segments, or, one wire segment to another. Logic circuits are implemented in FPGAs by partitioning the logic into individual logic blocks and then interconnecting the blocks accordingly via the switches.

4.3. FPGA Technologies

Manufacturers use many different technologies to create a wide variety of programmable logic devices (PLD) such as FPGAs. This wide variety of choices allows the developer to find a programmable device that best suits the particular

design without any difficulty. This report will discuss different FPGA architectures like Xilinx, Altera, Lucent, Actel, and Quicklogic to analyse the tradeoff between speed performance and area efficiency of a bit serial CORDIC chip.

4.3.1. Xilinx XC4000 Family

The basic building block of the Xilinx XC4000 family is called a Configurable Logic Block (CLB). Each CLB uses a two stage arrangement of lookup

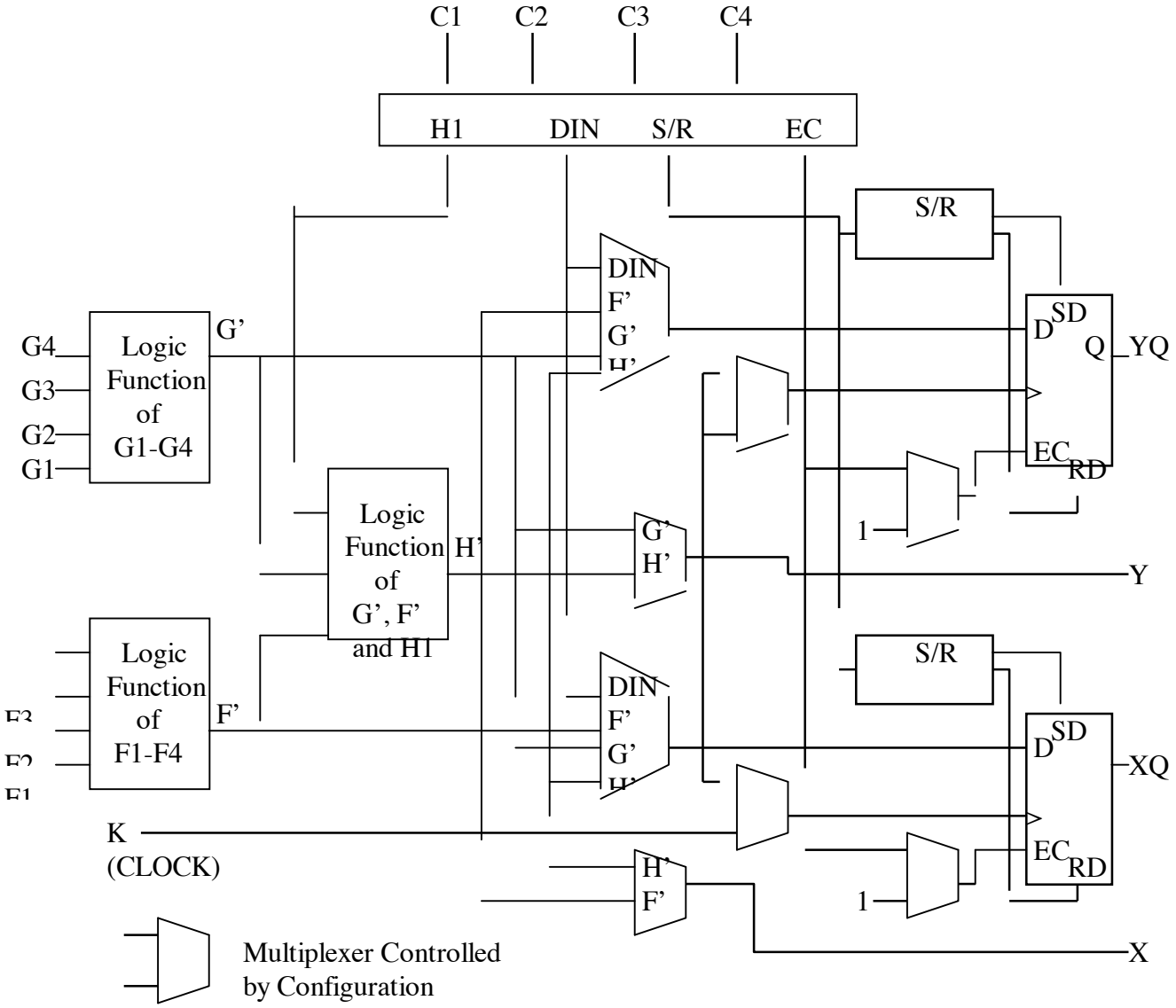

Figure 4.2 Block Diagram of XC4000 Series CLB [15]

tables to implement the logic functions, as illustrated in Figure 4.2. Each CLB is capable of implementing "any two independent functions of four variables, any single function of five variables, any function of four variables together with some functions of five variables, or some functions of up to nine variables" [13]. Dedicated arithmetic logic is also included in each CLB to speed up the generation of carry and borrow signals.

Each CLB has two outputs, each of which can be buffered. The buffers can be clocked on the rising or falling clock edge. The high number of buffers offered in the Xilinx chip makes it much easier top create synchronous designs. Synchronous designs are much easier to debug than asynchronous designs. Also, the high number of buffers makes it possible to create pipelined designs.

The performance of a system can be greatly improved by pipelining. This is because a design can be broken into smaller parts that can be executed in parallel. The results of each stage can be passed through the pipeline buffers to the next stage.

Xilinx uses Static Ram technology to control the multiplexers and transmission gates that are used to create the connections within the chip. The CLBs are placed in a two-dimensional array with routing connections made along the rows and columns of blocks. The different chips in the XC4000 family use varying sizes of CLB grids. The XC4052 uses 44x44 CLBs to connect together 1936 CLBs. The XC4052 has an equivalent gate count of 52,000 gates. The actual maximum number of gates that can

be implemented in a chip will vary from design to design, depending upon how well the design was mapped into the chip and how well the routing resources were utilized to make maximum use of the chip.

A Xilinx FPGA must be reprogrammed from an external source every time the chip is powered up. The data stream that is used to program the chip is created by a compiler program that receives the user's design. The program is responsible for deciding how to place the design within the chip and how to route the interconnection between the CLBs and within each CLB. How efficiently the resources of the chip are utilized are dependent upon how efficient the compile program is.

4.3.2. Lucent ORCA Series 3 Family

ORCA Series 3 FPGA consists of three basic elements called Programmable Logic Cells (PLCs), Programmable Input/Output Cells (PICs) and system level features. Each PLC contains a Programmable Function Unit (PFU), a Supplemental Logic and Interconnect Cell (SLIC), local routing resources and configuration RAM. Figure4.3 shows an array of PLCs surrounded by PICs. InterQuad routing blocks (hlQ, vlQ) present in series 3, are also shown. System level functions (located in the corners of the array) and the routing resources and the configuration are not shown in Figure4.3.

Most of the FPGA logic is performed in the Programmable Function Unit (PFU). Each PFU within a PLC contains eight 4-Input (16-bit) LUTs, eight latches/flipflops and one additional flip-flop that may be used independently or with

arithmetic functions. The PFU is organized in a twin-quad fashion: two sets of four LUTs and FFs that can be controlled independently. LUTs may also be combined for

Figure 4.3 OR3C/T55 Array [16]

use in arithmetic functions using fast-carry chain logic in either 4-bit or 8-bit modes. The carry-out of either mode can be registered in the ninth FF for pipelining.

The LUTs twin-quad architecture provides a configurable medium/large grain architecture that can be used to implement from one to eight independent combinatorial logic functions using multiple LUTs. The flexibility of the LUT to handle wide input functions, as well as multiple smaller input functions, maximizes the gate count per PFUwhile increasing system speed. There are different chips in ORCA series 3 family of different sizes of PLCs. The OR3T125 chip uses an array size of 28x28 PLCs put together 784 PLCs. The OR3T125 has an equivalent gate capacity of 92k - 186k gates. The number of PLCs or gates implemented in a chip varies from design to design. ORCA Series 3 devices contain many new patented architectural enhancements and are offered in a variety of packages, speed grades, and temperature ranges.

4.3.3. Actel 54SX Family

The basic architecture of Actel FPGAs consists of rows of programmable blocks called Logic Modules (LMs) with horizontal routing channels between the rows. SX family architecture has been called a “sea-of-modules” architecture because the entire floor of the devices is covered with a grid of LMs with virtuallu no chip area

lost to interconnect elements or routing. Each routing switch in these FPGAs is implemented by Anti-fuse programming technology.

Actel provides two types of Logic Modules, the Combinatorial Cell (C-cell) and the Register Cell (R-cell), each optimized for fast and efficient mapping of synthesized logic functions. The R-cell registers, as shown in Figure 4.4, feature programmable clock polarity, selectable on a register-by-register basis. This provides the designer with additional flexibility while allowing mapping of synthesized functions into SX FPGA. The clock source for the R-cell can be chosen from the hard-wired clock or the routed clock. The C-cell, as shown in Figure 4.5, implements a range of combinatorial functions

Figure 4.4 Logic Diagram for R-Cell [17]

up to 5-inputs. Inclusion of the DB input and its associated inverter function dramatically increases the number of combinatorial functions which can be implemented in a single module to more than 4,000 options in the SX architecture.

Actel has arranged all C-cell and R-cell logic modules into horizontal banks called Clusters. Actel has further organized these modules into SuperClusters to increase design efficiency and device performance. These clusters can be connected through new routing resources called FastConnect and DirectConnect which reduces the number of antifuses required to complete a circuit, ensuring the highest possible performance. The different chips in Actel uses varying sizes of gate capacity. The A54SX32 has a gate capacity of 32,000 gates and has maximum flipflops of 1980.

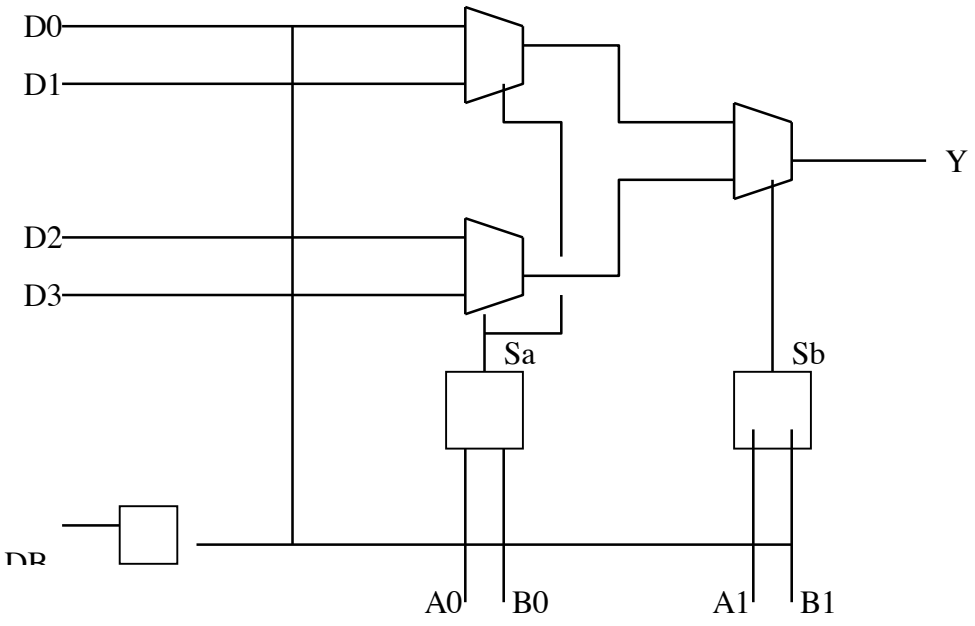

Figure 4.5 Logic Diagram for C-Cell [17]

4.3.4. Altera FLEX 10KE Family

Altera FPGAs are considerably different from the others discussed so far because they represent a hierarchical grouping of programmable logic devices. Nonetheless, they are FPGAs since they employ a two-dimensional array of programmable blocks and a programmable routing structure and are user-programmable.

Altera FLEX 10KE devices are an enhancement of FLEX 10K device family. The Flexible Logic Element Matrix (FLEX) architecture is based on reconfigurable SRAM elements. It incorporates all features necessary to implement common gate array mega functions. Each FLEX 10KE device contains an enhanced embedded array to implement memory and specialized logic functions, and a logic array to implement general logic. The embedded array consists of a series of Embedded Array Blocks (EABs) and the logic array consists of Logic Array Blocks (LABs). Each LAB contains eight Logic Elements (LEs) and a local interconnect.

Figure 4.6 shows a block diagram of the FLEX 10KE architecture. Each group of LEs is combined into an LAB and these LABs are arranged into rows and columns. Each row also contains a single EAB. The LABs and EABs are interconnected by the FastTrack Interconnect routing structure.I/O Elements (IOEs) are located at the end of each row and column of the FastTrack Interconnect routing structure. The FLEX 10KE family provides the density of upto 2,00,000 gates, speed and features to integrate entire systems, including multiple 32-bit buses onto a single device. The EPF10K100E device has 9 EABs and 624 LABs with Logic Elements of upto 4,992.

Figure 4.6 FLEX 10KE Device Block Diagram [18]

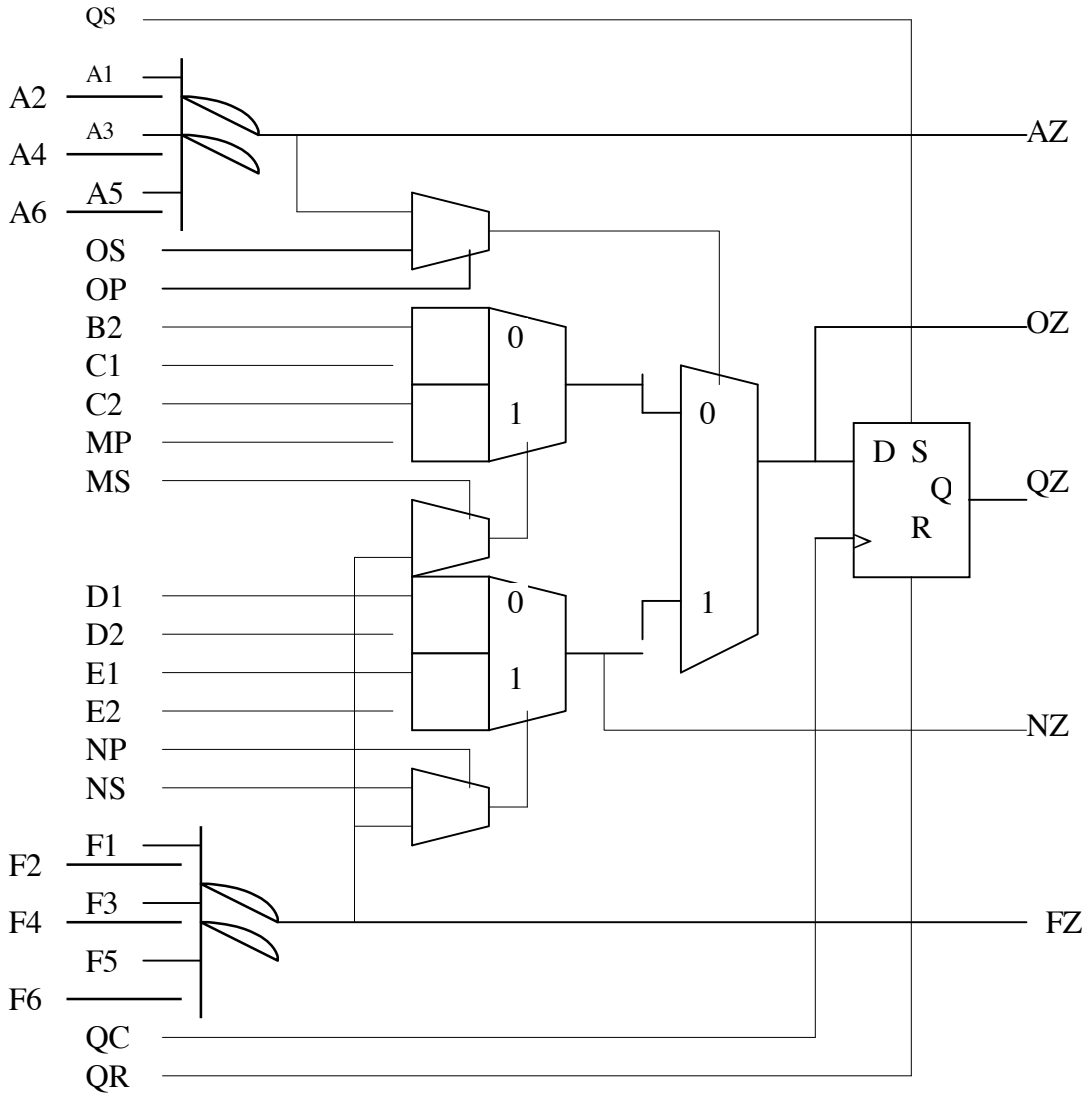

Figure 4.7 QuickLogic Logic Block [14]

The EPF10K100E device has an equivalent gate capacity of 1.00.000gates. The gate capacity of the chip will vary from design to design.

4.3.5. QuickLogic pASIC 3 Family

The pASIC 3 family uses QuickLogic's patented ViaLink technology to provide unique combination of high performance, high density, low cost, and complete flexibility. These devices are based on an array of highly flexible logic cells which have been optimized to efficiently implement a wide range of logic functions at high speed. Each logic cell includes one pre-configured register, plus the logic to implement an additional independent latch. Logic cells and RAM modules are configured and interconnected by rows and columns of routing metal and Vialink metal-to-metal antifuses.

The pASIC 3 logic cell shown in Figure 4.7 is a general purpose building block that can implement most TTL and gate array macro library functions. The cell has been optimized to maintain the inherent speed advantage of the Vialink technology while ensuring maximum logic flexibility. The function of a logic cell is determined by the logic levels applied to the inputs of the AND gates and multiplexers. Vialink sites located on signal wires to the gate inputs perform the dual role of configuring the logic function of a cell and establishing connections between cells.

The QL3040 is a 40,000 usable PLD gate member of the pASIC 3 family of FPGAs. The QL3040 contains 1,008 logic cells with a maximum of 252 I/Os. Software support for the complete pASIC 3 family is available through three basic packages, the turnkey QuickWorks, QuickChip and QuickTools which provides

complete FPGA software solution from design entry to logic synthesis, to place and route, to simulation.

4.4. Hardware Partitioning

When developing an algorithm for hardware implementation, there are considerations to keep in mind that depend upon the type of hardware that will be used. An efficient design for discrete logic may be very different than a design for a custom fabricated chip even though the two designs perform the same task. This is especially true when using FPGAs.

For this reason it is important to give careful consideration to the approach that will be used to implement the design. A well chosen approach will minimize the effects of the limitations imposed by the FPGA while still meeting the project's specifications.

4.4.1. FPGA Partitioning Considerations

An FPGA puts limitations on a design because of its physical construction. A particular chip has only so many available "gates" and I/O pins. It also has a maximum finite speed at which it can operate. Specific limitations will vary depending upon the chip to be used. The cost of the chip must also be kept in mind.

An FPGA has a maximum number of "logic gates" that it can represent at one time. The actual number will vary depending upon the design and how efficiently the compiler software maps the design into the FPGA. Manufacturers provide estimates of gate usage for different types of logic hat might be used in a design. These estimates cab be used to make rough estimates about the size of a preliminary design.

The interconnects used to move data from one logic block to another can also affect the size of a design. It is possible to fit a design into a chip, but not be able to connect all of the parts because there are not enough routing resources in a chip. For this reason, it is important to allow extra room in the chip. The fuller a chip is packed with logic gates, the more likely it is that the compiler will not be able to create a program that can connect everything together.

As early in the design as possible, a rough estimate of the size of the design should be calculated. As more details are decided, the design size should be recalculated. The purpose of this type of strategy is to avoid wasting time and effort developing a design that has no hope of ever fitting into a particular chip. On the other hand, it is also not expedient to spend time making a design very small only to find out that a large portion of the chip is unused.

When developing a preliminary design, keep in mind the amount of data that will have to enter and leave the chip. An FPGA has only so many I/O pins, some of which are used for power and ground connections.

Selecting a chip that does not have enough I/O pins could disallow a design or could severely limit the speed and performance of a design. On the other hand a chip with many unused pins will probably cost more than a chip that has only a few unused pins. In the initial stages of a design, the developer should select a chip that is a little lager than the estimated size of the design. This will allow extra I/O pins and “logic gates” for underestimated or forgotten sections of a design.

When developing a design, a developer should also keep in mind the speed requirements of the project. This is important for two reasons: FPGAs have speed limitations, and speed requirements partially determine the type of system architecture that should be used for the design.

FPGAs have maximum speeds at which they can be operated. The speeds will vary depending upon the type of chip that is used. The speed can also vary depending upon the complexity of design that is put into the chip. These factors must be considered when developing a design because they will partially determine how fast the design can run.

Different system architectures or system configurations can greatly affect the speed or performance of a design. Often the selection of an architecture is a tradeoff of speed and design size which is analyzed in this thesis.

Another topic to consider early in the design cycle is if the design is to be developed using a single chip or multiple chips. The major consideration here is cost. Using a single chip design, if possible, is usually much simpler to incorporate into a larger system. However, sometimes it is cheaper to use multiple smaller FPGAs to implement the same design. The decision can be made by comparing the cost of the system to how well it will meet the project specifications.

5. SNR Estimation Using Phase - Only Data

5.1. Motivation

"In communication systems, the received signal is usually a superposition of the transmitted signal and additive noise" [26]. Consider a digital receiver that receives a signal from two different receiving antennas separated apart at a certain distance. However, by the time the signal is received, noise and interference creates a phase margin between two signals leading to phase noise. This phase noise is calculated using the vectoring mode of CORDIC algorithm. As mentioned earlier in section 3.3.2, the vectoring mode is used to compute a rotation angle between the real and imaginary components of the vector.

The technique described in this chapter is used to develop a SNR estimation circuit which is designed and implemented in FPGA and is discussed later in this report. This chapter discusses a technique for calculating phase error and signal to noise ratio.

5.2. Digital Receiver

For a digital communication system, performance is generally measured in terms of probability of bit error (P_e) or equivalently, bit error rate (BER). P_e can be translated into SNR (section 4.3) when it is desirable to do so [10]. "Information is generally conveyed by means of a band pass signal, resulting from modulating a sinusoidal carrier"[11]. This signal can be represented as a sinusoid whose amplitude

and phase varies with time. The band pass signal x(t), represented in complex form, is given by

$$x(t) = \sqrt{2}\,\mathrm{Re}\,[x_L(t)\exp(jw_0t)]$$

This is called the complex baseband representation of a baseband signal, where $|x_L(t)|$ and $\arg(x_L(t))$ denotes the instantaneous amplitude and phase respectively, of the signal x(t). Expressing $x_L(t)$ in rectangular form yields

$$x_L(t) = x_c(t) + jx_s(t)$$

where $x_c(t)$ and $x_s(t)$ are the real and imaginary part of $x_L(t)$. They are also called as in-phase and quadrature components respectively with respect to an unmodulated carrier $\sqrt{2}\cos w_0t$.

5.3. Theory and Analysis of SNR technique

This section discusses a technique to calculate the SNR of the probability distribution of the phase of a sinusoid in the presence of narrow band additive white Gaussian noise. It is assumed that the distribution for the phase error is Guassian distribution.

5.3.1. Calculating the Phase Margin

The actual phase, ϕ, across any baseline of an interferometer is a linear function of the sine of the angle, θ, off the boresight. The slope, m, of the line is given by

$$m = 2\pi d/\lambda$$

where d is the length of the baseline and λ is the wavelength of the incoming signal. Let

$$t = \sin\theta$$

then

$$\phi = m * t.$$

The measured phase, ϕ, across the baseline is related to the actual phase by

$$\phi = \phi - 2\pi k + \epsilon$$

where ϵ is the error in the measurement and

$$k = [\phi / 2\pi]$$

where [x] is the nearest integer to x.

Ambiguities are defined as erroneous estimates of ϕ due to erroneous estimates of k.

For a linear array, the phases measured across each baseline can be plotted as an n dimensional function of t. This function will consist of a set of parallel lines in n-space. The lines are parallel because the slopes remain the same. The end points of the lines will all contain at least one element with a value of either π or $-\pi$. The end point of one line will corresponds to the end point of some other line by having the same value for ordinates except those which have value π or $-\pi$. This 'wrap around' effect results in an n-dimensional torus containing all the lines. The ambiguity rate is determined by the minimum distance between any lines on the torus.

Consider a three element array as an example. The distance between the baselines is easily calculated for most lines, however, the distance between lines A and B must be calculated by 'shifting' B from the corner ($-\pi$, π) to beyond the corner (π, $-\pi$). This distance is the actual distance between A and B. The points of boundary crossings (the points having at least one ordinate with value π or $-\pi$) are found using

the linear equation in t. The values of t associated with a boundary crossing are given by

$$t_{ij} = (2*I+1) * \pi/m_j$$

where m_j is the slope associated with the jth baseline. The point, x^{ij}, associated with the crossing is then given by

$$x_k^{ij} = t_{ij} * m_k.$$

The distance between lines can be calculated by rotating each line until they are parallel to one of the two axes. The lines can be thought of as points in n-1 space by considering only the intercept points of the line with any 'plane' perpendicular to the axis. Then, the distance between the lines is equal to the distance between the points.

Another method for calculating distances is to take any point on each line, calculate the length of the vector between points on different lines, and subtract off the length of the projection of that vector onto either line. The distance, d_{ijkl}, is given by

$$d_{ijkl} = \{ \| x^{ij} - x^{kl} \|^2 - [(x^{ij} - x^{kl}) . m]^2 / \| m \|^2 \}^{1/2}$$

where m is the vector containing the slopes. To account for the distance on a torus, phantom line segments are added to the set of line segments by adding or subtracting 2π from each element of each crossing point, yielding phantom crossing points. The phase margin is given by the minimum distance (greater than 0) between two lines, phantom or real.

This technique can easily be adjusted to account for correlation between the measurements. Let C be the variance-covariance matrix associated with the phase difference measurements. Then,

$$d_{ijkl} = \{ (x^{ij} - x^{kl})' c^{-1} (x^{ij} - x^{kl}) - (x^{ij} - x^{kl})' c^{-1} m/m' c - 1m \}^{1/2}.$$

5.3.2. Estimating the Ambiguity Rate

Given the minimum distance between lines, called the phase margin, for an antenna configuration, the ambiguity rate is found by assuming a distribution for the phase error and calculating the probability of the error in the phase difference across each baseline exceeding half the phase margin. Usually, the distribution assumed for the phase is the normal or gaussian distribution. By assuming that the error on each phase difference between elements is identically and independently distributed normal with mean, 0, and standard deviation, $\square^2$, the ambiguity rate, R, is given by

$$R = 2 - 2 * P [\square\square / \square < z]$$

where $\square\square$ is the phase margin. If the errors are biased with a random, non removable bias, the standard deviation of the bias is added to the standard deviation of the random error by taking the square root of the sum of squares of the standard deviations.

If the errors in the phase are not independent, as is often the case with baselines which share elements, the distance between lines must be adjusted to account for the covariance terms. Typically, this dependence will result in larger ambiguity rates. However, for arrays larger than 3 elements, the correlation could be used to improve the ambiguity rate by forcing the axes between the nearest lines to be perpendicular to the direction of the semi-major axes of the ellipsoid defined by the variance-covariance matrix. However, this technique will result in the lines becoming

parallel to the semi-major axis, resulting in larger AOA errors when no ambiguities are present.

5.3.3. Signal to Noise Ratios and Phase Error

The major source of phase error is usually the phase detectors. Since each baseline has its own phase detector, the errors introduced by the phase detectors are independent. When the phase detector errors dominate the total error, the errors in the phase differences can be considered as independent. The standard deviation of the phase difference is dependent only on the signal to noise ratio.

Given gaussian errors on the amplitudes of the sine and cosine of the incoming wave, then for a single channel, the error in the phase error is distributed by

$$f_{\Phi}(\Phi) = e^{-snr} \quad \Phi snr * \cos(\Phi) * \exp(-snr * \sin^2(\Phi))$$

where snr is the signal to noise ratio, and erf is the error function defined by

The density function is defined from $-\Phi$ to Φ.

The probability density function, $f_{\Phi}(\Phi)$, can be derived by integrating out the signal envelope term from the joint probability density function for the envelope and

phase of a sinusoid in the presence of narrow band additive gaussian white noise given by

Where the signal to noise ratio, snr, is expressed by

The expression to evaluate to obtain the phase error pdf is

The first step in evaluating this integral is to complete the square of the exponent in the exponential term

$$r^2 - 2 A_c r \cos(\theta) = [r - A_c \cos(\theta)]^2 - A_c^2 \cos^2(\theta)$$

Substituting this term into the integral yields

After

At this point the following substitution is made in the integral :

Using this substitution and performing some simplification, the integral becomes

This in turn may be manipulated to obtain

The integral

By making the substitution of $w = -y$ in the remaining integral term, $f_{\theta}(\theta)$, is rewritten as

Let $x = \sqrt{snr} * \sin(\theta)$, then

with x going from $-\sqrt{snr}$ to $\sqrt{snr}$. The first term is found by realizing that

The second term is found by realizing that

$$\sqrt{snr} * \cos(\theta) * \exp(-snr * \sin^2(\theta)) = -\sqrt{snr} * \cos(\theta - \pi) * \exp(-snr * \sin^2(\theta - \pi))$$

and

$$\sqrt{snr} * \cos(\theta) * \exp(-snr * \sin^2(\theta) * \mathrm{erf}\{\sqrt{snr} * \cos(\theta)\}$$

$$= \sqrt{snr} * \cos(\theta - \pi) * \exp(-snr * \sin^2(\theta - \pi) * \mathrm{erf}\{\sqrt{snr} * \cos(\theta - \pi)\}.$$

For large signal to noise ratios, the integral of the first term becomes very small ($4.5 * 10^{-5}$ at 10 dB). The value for erf (z) is greater than 0.9 for z greater than 1.17. Let

$$w = \sqrt{(snr - x^2)}.$$

Then, for a signal to noise ratio of 10dB,

$$\mathrm{erf}\,(w) \approx 1$$

for $\theta\theta\theta < 1.19$ (68^0) or $\theta\theta\theta > 1.95$ (112^0). Further, x is large whenever erf (w) is not near 1. Thus, the value of $f_\theta(\theta)$ is small for the region where value of erf (w) is not close to 1. If we approximate erf (w) by 1, and approximate the first term by 0, then

The limits on x are -θsnr to θsnr, which are large compared to range of values where f (x) is significant. That is

Thus x is approximately distributed normal with mean, 0, and variance (standard deviation squared), 1/2.

Since $f_\theta(\theta)$ is small for θ not near 0,

$$x = \theta\mathrm{snr} * \theta .$$

Thus, θ is approximately distributed normal with mean, 0, and variance, 1/ (2*snr).

Signal to Noise Ratio (dB)

Figure 5.1 Phase Error Standard Deviation True compared to Approximate

5.3.4. Conclusions

Some conclusions are drawn from the above derivation. The phase error for large signal to noise ratio can be approximated by the reciprocal of the square root of the signal to noise ratio. Figure 5.1 on the next page shows the phase standard deviation in relationship with signal to noise ratio. It is observed from the graph that the true value is almost equal to the approximated value for large signal to noise ratios and there is a significant variation in small signal to noise ratios.

5.4. Review

An SNR estimation technique used to calculate signal to noise ratio using phase -only data is derived and discussed in this chapter. True value of phase error is considered in this thesis to calculate small and large signal to noise ratios.

6. Implementation and Results

One way to deal with the problem of the increasing time-to-market pressures is to design at high levels of abstraction. Traditional capture-and-simulate methods have largely been given away to the describe-and-synthesize approach for these reasons. Hardware Description Languages (HDLs) have played an important role in the design synthesis methodology. They provide a means of specifying a digital system at a wide range of levels of abstraction. They are used for specification, simulation and synthesis of an electronic system. Among several different hardware description languages, VHSIC Hardware Description Language (*VHDL*) is one that is in widespread use today. It supports behavior specification at the early stages of a design process and structural specification at the later implementation stages. This chapter gives the detailed description of speed versus area tradeoffs in CORDIC processor and FPGA implementation for estimating Signal to Noise Ratio using CORDIC processor.

6.1. CORDIC Tradeoff

There are several ways that CORDIC processor can be implemented. An ideal way is to analyze speed versus area tradeoffs by mapping into different FPGA architectures. CORDIC processor can be designed using bit-serial or bit-parallel arithmetic. However, bit-serial arithmetic provides relatively compact design and works faster than bit-parallel design because of its simplified interconnect and logic

[6]. “The bit-parallel variable shift registers do not map well to FPGA architectures because of the high fan-in required” [6].

6.1.1. Bit-Serial CORDIC design

The bit serial CORDIC processor is shown in figure 6.1. In this design, *p* clocks are required for each of the *n* iterations, where *p* is precision of the adders. The precision used in this project is upto 17bits and the iterations are limited to 8. “At the beginning of each iteration, the control state machine reads the sign of the y (or z) register and

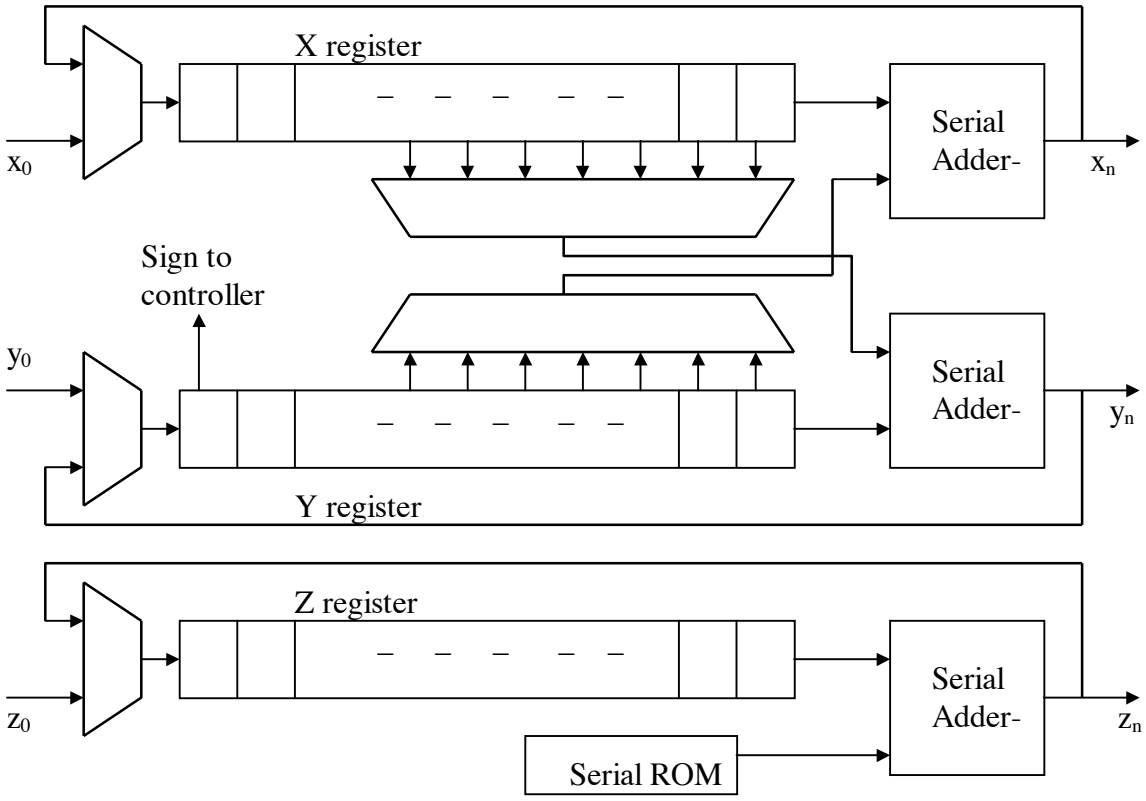

Figure 6.1. Bit Serial iterative CORDIC [6]

sets the add/subtract controls accordingly. The appropriate cross terms are also selected at the beginning of each iteration" [6]. During the 8th iteration, the results can be read from the output pins of the serial adder/subtractors.

6.1.2. Simulations

As a first step to understand this algorithm, it is often desirable to simulate the process in software. Such is the case with the CORDIC algorithm. Simulation at this level can be done early in the design process. This is desirable because it allows the designer to try different methods of implementing a design without having to design and debug actual hardware. VHDL is designed to fill a number of needs in the design process. It allows the design of a system to be simulated before being manufactured so that designers can quickly find errors and correct them.

The simulation for this project is accomplished in two stages. The first stage is the implementation of the design of bit-serial CORDIC processor. The second stage involves the estimation of the Signal to Noise Ratio (SNR) using phase-only data where the CORDIC processor is used to find the angle made by the incoming complex signal that is received by a digital receiver. The source code for the entire design is included in this report in Appendix A. A VHDL simulator tool called *QuickVHDL Mentor Graphics* is used in this project to implement the design.

6.1.3. Logic synthesis for Field Programmable Gate Arrays

Logic synthesis takes the circuit description at the Boolean level and generates an implementation in terms if an interconnection of logic gates. Typically, synthesis is done for an objective function, such as minimizing the cost of the design satisfying the

performance constraints, minimizing the power consumed, or making the implementation more testable. The cost of the design may be measured by the area occupied by the logic gates and the interconnect.

Since synthesis is a difficult process, it is seperated into two phases. The first phase is techology-independent optimization phase, which generates an optimum abstract representation of the circuit. The second phase is technology mapping phase in which the optimized representation is mapped onto a pre-defined library of gates.

Various synthesis software tools are in use today. *Synplify* is one of the logic synthesis engine that utilizes proprietary Behavior Extracting Synthesis Technology (B.E.S.T.) is designed to deliver fast, highly efficient FPGA and CPLD designs. *Synplify* takes *Verilog* and *VHDL* Hardware Description Languages as input and outputs an optimized netlist in most popular FPGA vendor formats. This software tool was used to map VHDL design to different FPGA architectures.

6.1.4. Speed versus Area Tradeoff

The total chip area needed for an FPGA consists of the logic block area plus the routing area. Since routing area takes around 70 to 90 percent of the total area, the effect of logic block functionality on the routing area can be very important. Also, routing area depends on the type of programming technology used for programming resources and routing switches.

Currently, some of the highest density FPGAs are built using Static Memory (SRAM) technology. The other common process technology is called Antifuse which has benefits for more plentiful programmable interconnect. SRAM based FPGAs are

in-system programmable (ISP) which means that the device can be programmed while it is mounted on the board with other components, where as Antifuse based FPGAs are one-time programmable (OTP) which means that once programmed, they cannot be modified. But, they also retain their program when the power is off.

On the other hand, the performance of the system depends on the size of the lookup tables (LUTs). Larger LUTs cover more gates but usually have longer delay. Also, "the performance diminishes with the increase in word width because of the carry propagation delay across the adders" [6].

Device Type	**Estimated Frequency (MHz)**	**% chip Area Required**
Xilinx XC4052XL	**20.5**	**78%**
Altera FLEX10K100E	**13.1**	**82%**
Lucent ORCA3T125	**11.8**	**60%**
Actel A54SX32	**16.6**	**165%**

Table 6.1 Tradeoff Results

After developing the design of Bit-Serial CORDIC, it is very easy and convenient to map the VHDL design to different FPGA architectures such as Xilinx, Altera, Actel, and Lucent to analyze speed versus area tradeoff. Table 6.1 illustrates the results of speed and area of various device families. Current investigations show that Xilinx, Altera and Lucent require some part of the total FPGA chip, whereas, Actel needs more than one chip which is not a very appreciable result. Xilinx FPGA requires larger area than Lucent FPGA, but lesser than Altera. On the other hand, Xilinx runs at higher frequency than other FPGA families. So, there is a tradeoff.

6.2. Block Diagram

As discussed before, the CORDIC processor is used in various applications. One such application that is discussed in this project is estimating Signal to Noise Ratio (SNR) using phase-only data. The vectoring mode of CORDIC rotator is used in calculating the phase of the incoming signal from digital receiver.

Figure 6.2 shows the block diagram of SNR estimator. The angles made by the corresponding complex signals that are received by a digital receiver are calculated by the CORDIC rotator. Since the angles from CORDIC are represented in BAMs, they are converted to Radians in Binary representation by ROM. Then, the phase subtractor outputs the phase margin leading to phase noise. The variance is calculated for an average of 128 samples and finally the ROM outputs the SNR for the corresponding variance.

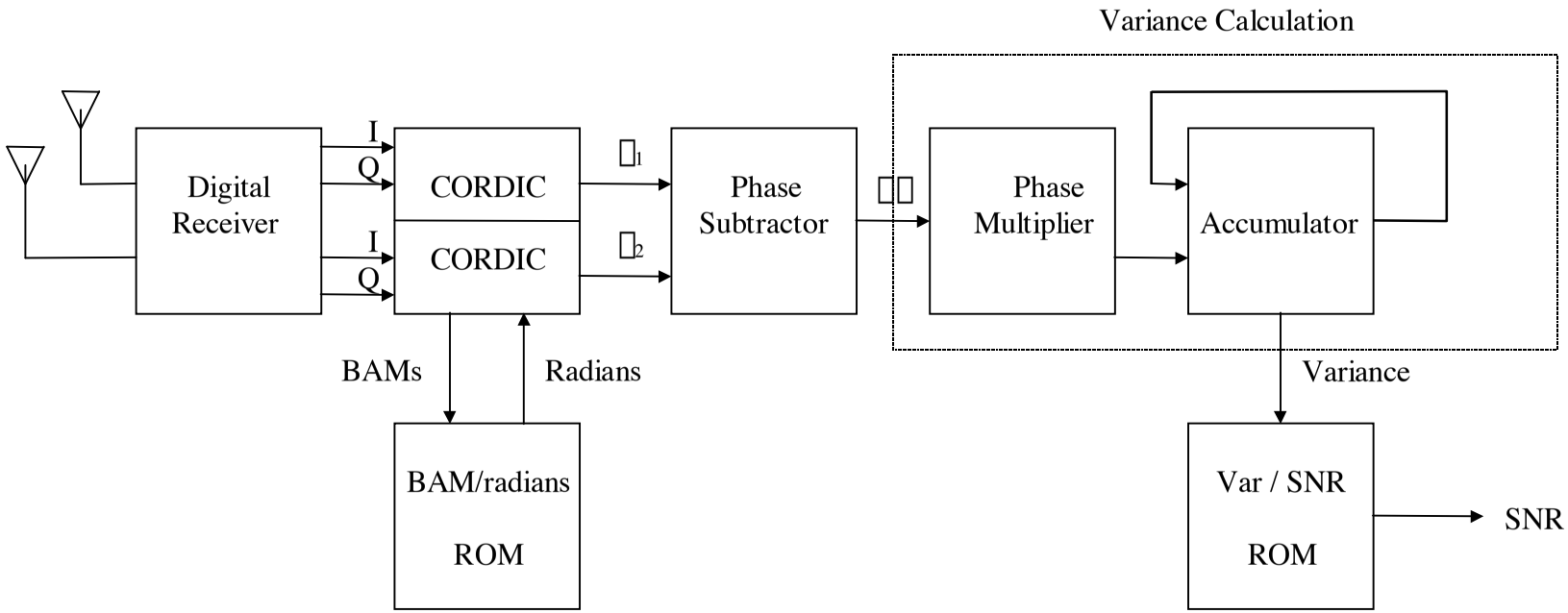

Figure 6.2. SNR Estimator

6.3. SNR Estimation

Basically there are two approaches to SNR estimation at the receiver of a communication system: decision-aided estimation and blind estimation. Decision-aided estimators use estimated data to obtain an estimation of the SNR. While this approach works fine at low data error rates, it shows high estimation variance when the reliability of the decoded data is low.

The blind estimator investigated here uses purely statistical knowledge about the (undisturbed) signal and the expected noise (e.g. Gaussian distribution is assumed). The SNR is estimated by a simple algorithm that calculates the variance of the probability distribution of the phase of a sinusoid in the presence of narrow band additive white Gaussian noise.

The block diagram showed in Figure 6.2 is implemented in VHDL. This design accepts 256 vectors and each time the receiver receives two vectors, calculates the phase difference and the resulting sample is stored in the accumulator. This process is repeated for 256 vectors and the variance is calculated for an average of 128 samples. This whole process is pipelined and is implemented in VHDL. Table 6.2 illustrates the results of True SNR and Circuit SNR calculations. Theoretically, True SNR should be equal to Circuit SNR. However, in the practical sense, the SNR will not be exactly equal. The maximum error observed in SNR from the results is 0.31db.

True Variance (rad)	True SNR (db)	Circuit Variance (rad)	Circuit SNR (db)
0.7447214	0.075	0.7447253	0.0751
0.0823103	8.255	0.0823792	8.261
0.00347123	21.6	0.00348175	21.66
0.02105528	13.85	0.02110275	13.88
0.000505058	29.93	0.00051033	30.24
0.2349831	4.565	0.23529195	4.571
0.007163211	18.47	0.00717484	18.5
0.005190848	19.86	0.00519895	19.892

Table 6.2 SNR Estimation Results

After implementing the SNR design in VHDL, the entire design is mapped into Xilinx FPGA technology to develop a prototype SNR estimation circuit. The device type used here is Xilinx XC4052XL. The total packed CLB's available in this device type are 1936. It is observed from the synthesis results that the total packed CLB's required for this circuit are 3152. These results shows that the SNR estimation circuit requires an area of 163% (i.e., more than one chip). The Throughput of this circuit is estimated to be around 9765 SNR's/sec.

7. Conclusion

7.1. Thesis Results

This thesis presents the background , design and simulation of a SNR estimation algorithm as an extended application of CORDIC processor. The CORDIC processor is a 17 bit, fixed-point representation of the CORDIC algorithm in a FPGA. The design analyses speed and area tradeoff between FPGA architectures. CORDIC processor calculates trigonometric, linear and hyperbolic functions and also computes polar to rectangular and rectangular to polar transformations.

The purpose of this project is to develop a FPGA based circuit which directly estimates the SNR of signals received by two different antennas at the reception of a digital receiver. This thesis calculates the signal to noise ratio using only phase data which is provided by CORDIC processor. This report discusses a technique for calculating phase error and signal to noise ratio.

The background information is intended to give the reader an understanding of the CORDIC algorithm and SNR estimation technique. Included in this background is a discussion of how the algorithm works, how it is applied to calculate signal to noise ratio and how the area and performance differs in different technologies and the tradeoff. Also included in the background is a discussion of the architectures of current

FPGA technologies and why FPGAs are becoming a powerful force in the field of digital hardware.

This project begins with the design and implementation of CORDIC algorithm in FPGA and investigating the speed and area in different FPGA technologies and analyzing the tradeoff. This design is used in estimating the signal to noise ratio using phase only data. SNR estimation is simulated in VHDL which makes very convenient to develop the hardware design. This process is completed when the design is successfully implemented in one of the FPGA technology.

7.2. Future Directions

Future work on the ideas presented in this thesis is predominantly in the area of redesigning the system. This thesis calculates SNR for an average of 128 samples. It is possible that the SNR estimation circuit cane be redesigned using 256 samples. Additional work could be done analyzing the area and performance tradeoff between iterative method and pipelined method of implementing the design.

Bibliography

[1] J. E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Trans. Electron. Comput.*, vol. EC-8, pp. 330-334, Sept. 1959.

[2] J. S. Walther, "A unified algorithm for elementary functions," *1971 Spring Joint Comput. Conf.*, AFIPS Proc., vol. 38, pp. 379-385.

[3] H. M. Ahmed, "Signal Processing Algorithms and Architectures," Ph.D. dissertation, Univ. of Stanford, June 1982.

[4] Yu Hen Hu, "The Quantization Effects of the CORDIC Algorithm", *IEEE Trans. Sig. Proc.*, vol. 40, no. 4, pp. 834-844, April 1992.

[5] Xiaobo Hu, Ronald G. Harber, and Steve C. Bass, "Expanding the Range of Convergence of the CORDIC Algorithm," *IEEE Trans. Comput.*, vol. 40, no. 1, pp. 13-21, January 1991.

[6] Ray Andraka, "A Survey of CORDIC Algorithms for FPGA based computers," FPGA '98. Proceedings of the 1998 ACM/SIGDA sixth international symposium on field programmable gate arrays, Feb. 22-24,1998, Monterey, CA, pp. 191-200.

[7] Raymond E. Fowkes, "Hardware Efficient Algorithms for Trigonometric Functions," *IEEE Trans. Comput.*, vol. 42, no. 2, pp. 235, February 1993.

[8] R. J. Andraka, "Building a High Performance Bit-Serial Processor in an FPGA," Proceedings of Design Supercon '96, pp. 5.1 - 5.21, January 1996.

[9] Rajeev Murgai, Robert K. Brayton, Alberto Sangovanni-vincentelli, Logic Synthesis for Field-Programmable Gate Arrays, Kluwer Academic Publishers, 1995.

[10] Jerry D. Gibson, Principles of Digital and Analog Communications, 2nd Edition, Newyork Macmillan, 1993.

[11] Heinrich Meyr, Digital Communication Receivers, Newyork Wiley, 1998.

[12] Andrew Wheeler, "Rapid Prototype of an SIMD Processor Array (using FPGAs)", Master's Thesis, University of Arkansas, 1993.

[13] Stephen D. Brown, Robert J. Francis, Jonathon Rose, Zvonko G. Vranesic, Field-Programmable Gate Arrays, Kluwer Academic Publishers, 1992.

[14] Ian Hickman, "Understanding Phase Noise," *Electronics World*, vol. 103, pp. 642-646, August 1997.

[15] The Programmable Logic Data Book, Xilinx, 1994.

[16] "ORCA Series 3 FPGAs Data Book," Lucent Technologies, http://www.lucent.com/micro/fpga/3ctxx.html (current Apr. 20, 1999).

[17] "Actel SX Data Sheet," Actel, http://www.actel.com/products/devices/SX/sxp.html (current Apr. 20, 1999).

[18] "Altera FLEX10KE Embedded Programmable Logic Family Data Sheet," Altera, http://www.altera.com/html/literature/lf10k.html (current Apr. 20, 1999).

[19] T.R. Benedict, T.T. Soong, "The joint estimation of signal and noise from the sum envelope," *IEEE Trans. Inform. Theory*, vol. IT-13, no. 3, pp. 447-454, 1967.

[20] B. Shah, S. Hinedi, "The split symbol moments SNR estimator in narrow-band channels," *IEEE Trans. Aerospace and Electronic Systems*, vol. 26, no. 5, pp. 737-747, 1990.

[21] D.R. Pauluzzi, N.C. Beaulieu, "A comparison of SNR estimation techniques in the AWGN channel," IEEE Symposium, Victoria, Canada, May 1995.

[22] Rolf Matzner, "An SNR estimation algorithm for complex baseband signals using higher order statistics," *Facta Universitatis (Nis)*, no. 6, pp. 41-52, 1993.

[23] Jean Duprat, Jean-Michel Muller, "The CORDIC Algorithm: New results for fast VLSI implementation," *IEEE Transactions on Computers*, vol. 42, pp. 168-178, 1993.

[24] C. E. Gilchriest, "Signal-to-noise monitoring," *Spacing Programs*

Summary 37-27, IV, Jet Propulsion Laboratory, Pasadena, CA, june 1967, pp. 169-184.

[25] J. W. Layland, "On S/N estimation," *Space Programs Summary* 37-48 III, Jet Propulsion Laboratory, Pasadena, CA, Dec. 1967, pp. 209-212.

[26] "SNR estimation", Institute of communications Engineering, http://speedy.et.unibw-muenchen.de/forsch/ut/moment/snr/index.html (current june 30, 1997).