WCDMA BASED RECEIVER DESIGN AND IMPLEMENTATION ON FPGA FOR SOFTWARE DEFINED RADIO

by

Waqas Mazhar

2010-NUST-MS PHD- ComE-06

MS-65

Submitted to the Department of Computer Engineering in fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

Thesis Supervisor

Prof. Dr. Shoab Ahmad Khan

College of Electrical & Mechanical Engineering

National University of Sciences & Technology

2013

DECLARATION

I hereby declare that I have developed this thesis entirely on the basis of my personal efforts under the sincere guidance of my supervisor Prof. Dr. Shoab Ahmad Khan. All of the sources used in this thesis have been cited and contents of this thesis have not been plagiarized. No portion of the work presented in this thesis has been submitted in support of any application for any other degree of qualification to this or any other university or institute of learning.

Waqas Mazhar

ACKNOWLEDGEMENTS

All praises to Allah Almighty, the most gracious the most merciful and the unprecedented Innovator of this universe, for giving me courage and leading me through. Without His guidance, it would not be possible for me to complete this landmark. Moreover, I am thankful to my parents for their prayers, encouragement and moral support in my entire academic career.

I am deeply thankful to my thesis supervisor, Dr. Shoab Ahmed Khan, for his guidance. Under the shadow of his command, I am able to accomplish this milestone. He is one of the very few people I have met in my life, who has inspired me to unimaginable scale and encouraged me to take challenges with interest. Moreover, I would also like to thank Mr. Muhammad Zeeshan Attari from Center for Advanced Research in Engineering (CARE) who helped me figuring out the specifications of the design to be implemented.

My compliments to the guidance committee members especially to Dr. Saad Rehman whose coaching in Computer Architecture and Parallel Processing has helped me a lot in the implementation of design. The valuable suggestions and guidance of Dr. Arslan Shaukat and Dr. Sheikh Farhan were great source of formatting my thesis and improved presentation of my research work.

My deep hearted thanks to long list of friends especially my roommates Mr. Murtaza Naqvi and Mr. Farhan Ahmed. They cooperated a lot throughout the session of this research work. My other colleagues such as Mr. Nadeem Iqbal, Major Salman Rasheed helped me wherever possible. I wish them a big good luck for their career.

Dedicated to Hard Working People of Pakistan

ABSTRACT

Software Defined Radio (SDR) is of big importance when it comes to secure communication in hostile environment. In the signal jamming conditions, the effects on the received signal are undesirable. To counter this problem, the signal to be transmitted is spreaded over the entire bandwidth using chip code. The receiver then uses the same code for despreading. Moreover, to counter the undesirable effects of channel, the training sequence is appended with data before transmission. The receiver uses this training sequence to equalize the effects caused by the channel. In this thesis, two parts of receiver of software defined radio have been implemented. It includes CORDIC and Coarse Frequency Estimation. Implementation has been realized on FPGA because communicational algorithms that provide respective solutions are computationally intensive. Chosen algorithms for implementation are well evolved and robust. Detailed design of each of the implementation has been presented along with description. Two designs are given for coarse frequency estimation algorithm; one of them is optimized for area and other is for performance. Moreover, the CORDIC implementation is optimized for both accuracy and area. Numerous fundamental principles of both signal processing and digital system design have also been mentioned as a part of literature review. Detailed results including percentage error, throughput and resource consumption are provided. In conclusion and future work, the parts of the SDR that can work in cascade with implementation are discussed.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1: Introduction

1.1 Overview

The Software Defined Radio (SDR) is one of the most emerging technologies in the field of wireless communication. The reason for which it is known as 'Software Defined' is the fact that most of the operations done on the signal like modulation, demodulation, frequency estimation, channel estimation are configurable and controllable by the software. For different SDRs the extent to which these operations are configurable by the software is different and purely defined by its architecture. Moreover, it also helps in facilitating the connectivity with wide range of wired protocols and also can be to designed to work equally well with standard wireless protocol implementations without changing baseline hardware.

Its major advantage over the conventional analog radios is reduced error rate and facility to encrypt the data sent over the wireless channel. The reason being the digitally designed components are more predictable and reconfigurable in fulfilling the required design characteristics and hence far more consistent and deterministic then their analog counterpart. The power of hardware is increasing rapidly with time which enables radio designers to shift more and more part of the radio in the software. So, as a bonus, in order to change the functionality, designers do not need to change the underlying hardware which reduces the overall development resource consumption.

Due to its ability to include multiple encrypting techniques for highly secured communication, it is very popular among military forces of many countries. Moreover, countries tend to use the legacy GSM systems in order to communicate at the long distances. For Example, In Pakistan, the commercial telecommunication network is very strong along the eastern border with India. SDR can be designed to work with GSM protocol with its own baseband encryption, making it possible to send messages to any part of a country.

For the communication with another local radio directly, it generally uses jamming resistant techniques. These techniques spread the transmitted signal over its entire bandwidth irrespective of the changes in baseband signal. Moreover, encryption is also employed on the baseband signal which makes it further secure. Typical abstract architecture of SDR is shown in Figure 1.1.

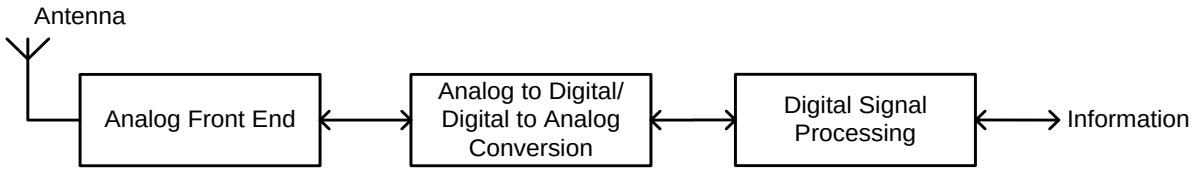

Figure 1.1 Abstract Architecture of Software Defined Radio

One of the most vital parts of SDR is analog front end. It converts the high frequency modulated signal to an intermediate frequency which digital system is able to process. This process is known as mixing. Analog to Digital Converter converts the analog signal at intermediate frequency to digital signal for Digital Signal Processor (DSP) to process. DSP is the most complex and most reconfigurable part. It processes the digitally converted signal for necessary algorithms of digital communication. This may include Start of Burst Detection, up or down sampling, intermediate carrier frequency estimation, channel equalization and many others. Information baseband signal is the output of DSP which is given to the type of output it belongs to. Reverse is the process in case of transmitting information.

1.2 Background

The specific SDR, for which research has been done in this thesis, is the project of Center for Advanced Research in Engineering (CARE), Islamabad and it is designed for fulfilling the portable communication needs of Pakistan Armed Forces. The hardware boards for these SDR are prepared by the Turkish based company Aselsan Radio. The major contribution of CARE, Islamabad is to design the brain of hardware board which includes the design of different architectural layers including physical and networking layer. The technology of implementation of architectural layers is hybrid. It includes embedded microprocessors, DSP processors and Field Programmable Gate Arrays (FPGAs).

Basic requirements of this SDR, being a part of military equipment, are that it must be highly jamming resistant, high level data encryption, automatic error correction of bit stream and high data rate. To meet all these requirements, the design of WCDMA at abstract level is shown in Figure 2.14. In this dissertation, the major concentration will be on the design of following constituents of receiver.

- Coarse Frequency Estimation
- CORDIC Implementation

1.3 Problem Description

In this section, the main objectives of design of parts of Software Defined Radio are given.

1.3.1. High Data Rate

Design of the parts of the SDR is required to achieve high data processing throughput. Mainly two designs of Coarse Frequency Estimation Algorithm have been implemented. One of them is targeted to the SDR of high throughput while other targets medium data rate SDR. High throughput corresponds to the SDR of throughput up to 384 kbps while medium throughput relates to the SDR of throughput up to 128 kbps. These implementations may give user the choice to choose one of the designs depending upon the resources available and targeted throughput.

Due to the complexity of the algorithms, the target technology for the receiver will be Field Programmable Gate Array. The reasons of the choice of target technology are given under topic Target Technology Selection on Page 16. Moreover design must be extendible so that it is possible to make it work for future configurations.

1.3.2. Demodulation

To demodulate the receiving signal, the exact frequency of the carrier is required to be multiplied with transmitted signal. The problem exists is the regeneration of the exact carrier signal at the receiving end. Easiest way of demodulation is by using crystal oscillator to regenerate an exact frequency of carrier at receiver end and multiply it with received signal. This type of

demodulator is known as synchronous or coherent demodulator [1]. For this type of demodulation, following problems will occur.

- Crystal oscillator of exactly same frequency must be employed for transmitter and receiver.
- With the process of aging, the crystal oscillator will start to divert from the frequency signal for which it was employed. This will cause distortion in the receiving signal.
- Overall cost of the design will increase.

The most feasible solution to this problem is to implement some method that can measure the frequency of the received carrier and generate that frequency. But alternative methods are more complex and require more resources to implement.

1.4 Target Technology Selection

The targeted technology for the implementation of physical layer is the baseline hardware on which the design is incorporated. Basic parameters that govern the selection of category of device technology for implementation are;

- Time to market.
- Maintainability.
- Area of specialization.
- Computational complexity of design.
- Power constraints on the system.
- Industrial Standards for targeted application.
- Testing Requirements.
- Accuracy in calculation.

. It may include one or multiple of the following technologies.

- DSP Microprocessors
- Micro Controllers
- FPGAs

Most of DSP Microprocessors and Microcontrollers have Harvard, Modified Harvard or Von Neumann Architecture. Von Neumann Architecture is explained in great detail in [2]. Major difference between these architectures is Von Neumann architecture has only one memory which is served as program and data memory. Harvard Architecture postulates of having separate memory both for data and program. This increases the speed of the architecture as program instruction and data can be obtained simultaneously. But Modified Harvard Architecture does not allow this strict partition. In this architecture, data can be accessed as an instruction and vice versa.

1.4.3. DSP Microprocessors

DSP Microprocessor is the type of processor that is specialized for DSP applications. They are generally Reduced Instruction Set Computer (RISC) based. Its instruction set includes instructions that can effectively perform DSP operations using hardware acceleration.

Advantages of DSP Processors are

- DSP microprocessors are optimized with respect to power consumption. They are also designed to consume power variably depending on the performance required by the application.
- Low time to market along with only software of DSP processor is required to be tested.
- Change in functionality would be achieved by changing the software only.
- Overall maintainability cost is quite low and computational complexity capability is high. Their overall computational capability and power consumption depends on the architecture and clock speed.
- Many high end DSP Processors have support for single precision floating point calculation in hardware. This allows the programmer to perform DSP operations with very high accuracy.

Weakness of DSP microprocessors is following

- If the implemented design in DSP does not meet performance requirements, it may be needed to replace it with higher benchmark DSP processor. It can cause design to change.

1.4.4. Microcontrollers

Microcontrollers are small computers that can be programmed to do a specific task. They have integrated resources that can be used to store programs. Microcontrollers are both RISC and Complex Instruction Set Computer (CISC). Both RISC and CISC are explained in [2] and [3]. Microcontrollers may have both fixed and floating point arithmetic implementation. Hence accuracy of their computation varies.

Advantages of Microcontrollers are

- They contain major resources on board. So, they can operate standalone. This decreases the cost for hardware design.
- Optimized for low power consumption.
- Low Time to market.
- High maintainability.
- To change functionality, only software needs to be changed.

Disadvantages of Microcontrollers are

- As many other resources are also fabricated along with the processor, the overall processing capability of processor is quite low. Microcontrollers are generally targeted for low end and highly power constraint applications.
- If required processing power is more than offered by a microcontroller, generally controller needs to be changed.

1.4.5. FPGAs

FPGA is a reprogrammable device that can be used to implement hardware of any digital design defined by a software code. The major vendors are Xilinx and Altera. Its technology is completely different from Application Specific Integrated Circuit (ASIC) in the sense that any digital hardware can be implemented and erased making it flexible to work with large number of applications. The basic functional unit for logic implementation is Logic Element (LE) in Altera and Slice in Xilinx FPGAs. Both of these units have multiplexers and Look up Tables (LUT) to

implement digital logic of any kind. These units are connected together with programmable connections.

Generally the design of FPGA is targeted for high throughput DSP and Networking Applications. Being very important in the field of DSP, designers of FPGAs have provided hardware acceleration of extensively used blocks. For example, as multiplication and accumulation is very widely used operation, their support is provided in hardware i.e. special dedicated blocks are made to take care for these operations.

Microprocessors, both of hard-core and soft core type are used in FPGAs. By hard core microprocessors, it means that processor is already fabricated within FPGA. Whether or not that processor is utilized, it always remains inside as an extra resource. These processors can be connected with custom logic developed inside FPGA logic units. Xilinx FPGAs have 32 bit version of hard-core Power PC microprocessor.

Soft-core processors are the Intellectual Properties (IP) of certain company that can be employed using the basic building blocks of FPGA. Xilinx provides 32 bit Microblaze and 8 bit Picoblaze microprocessors. Unlike soft-core, hard-core processors do not consume a single logic unit in an FPGA. Moreover, hard-core processors are faster than soft-core processors.

Major Advantages of FPGAs are

- Hardware with optimized performance for a specific purpose can be designed.
- Performance can be scaled by employing extra logic. Performance scaling is only limited by the area of FPGA.
- Unlike ASICs, hardware design can be changed according to the requirement.
- FPGA contains on chip memory that enhances the speed of design.
- Exact Requirement Specific design can be made at low cost while getting much higher performance than that of General Purpose Processors (GPP) and DSP Processors.
- Microprocessors can be connected with custom logic and that logic can be used by giving instructions in software. This looks like a processor with hardware acceleration of some certain set of instructions.

Major Disadvantages of FPGAs are

- In case of design change, hardware needs to be changed that means resources of hardware testing are needed.
- While development, software and hardware both needs to be tested. This requires a lot of extra effort.
- Power consumption of FPGA is higher than that of microprocessors and microcontrollers.
- Extra skills are needed to develop and test hardware in logic units of FPGA. It increases the cost of overall design [4].

1.4.6. Overall Comparison

Based on the analysis of problem description along with the strengths and weaknesses of nominated target technologies, FPGA is the best option for implementation. Briefly stated; algorithms to be implemented are highly computationally intensive and are not a part of any standard implementations, it is quite evident that there is a need to map entire application on FPGA for effective implementation [5].

Overall comparison is summarized in Figure 1.2.

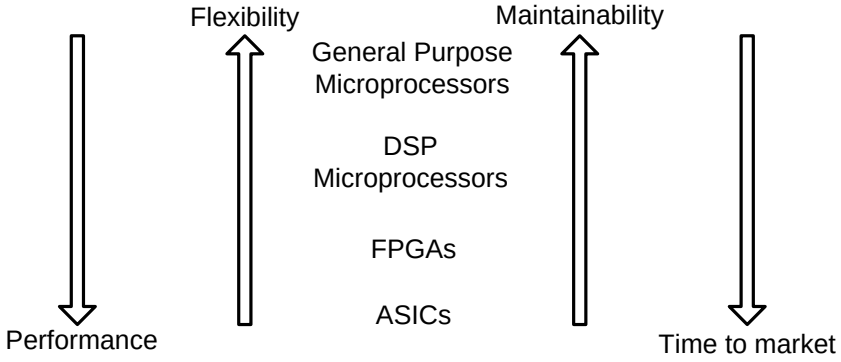

Figure 1.2 Target Technologies Comparison against multiple trends

1.5 Thesis Organization

This document includes the details of the overall information gathered for the research work along with the proposed design of parts of Software Defined Radio. Chapter 1 is the introduction and background to the research work. It is followed by the Chapter of Literature Review that

contains relevant researches and concepts. This chapter contains the details of primary concepts used in the implementation and proposed design. Moreover, it also comprises the previous relevant implementations and thesis related calculations.

Chapter 3 has detail of the proposed design of various parts of SDR; how and what was the approach of this design is the key concentration of it. All details are illustrated by the tables and figures of proposed design. It is followed by the Chapter 4 which is of results of the implementation of proposed design and its discussion. Final is the Chapter 5 which briefly concludes the whole research and presents the opportunities of the work that can be done in future.

Chapter 2: Literature Review

2.1 Introduction

This chapter summarizes the details of research in the field of SDR. A large number of research citations have been published in this accord. Although there are many citations, this literature review concentrates on the design theme of the physical layer of SDR. Design theme majorly includes the translation of the mathematical algorithms into architecture such that the requirements assumed for design are fulfilled. Analyses of algorithms are done and various other parameters are discussed that affect the design and its performance.

Already published citations include architectures that are optimized for different requirements. Moreover, algorithms used in these architectures are also discussed. Various competent technologies for SDR are mentioned and fact based analysis is done on technologies using which architectures can be implemented and reasons are given on why a particular technology is selected. The overall abstract design of the SDR for which the research has been conducted is also given.

This chapter starts with the discussion on the fundamental concepts related to digital communication and digital system design techniques with relevant examples. All of these topics also include relevant technical terms and their definitions. It is then followed by the trends in the research of digital system design and implementation of architecture in different publications. After providing further relevant information, this chapter concludes on the algorithms used in the design, mathematical calculation and abstract design of overall SDR used in this research.

2.2 Basic Concepts

This section contains all of the most basic relevant concepts used in the research. Most of these concepts are related to terminologies and definitions of digital communication in wireless domain and digital system design.

2.2.3. Digital Communication

When two devices communicate with each other digitally, such that most of the processing of data is done in digital domain, it is known as digital communication. By digitally, it is meant that a received signal is converted into digital stream before a receiving device can operate and find the message hidden in it. Reverse is the case for signal transmission; a device processes a message into digital stream and then that stream is converted into analog signal before transmission. This stream is converted into analog waveform before transmission. Digital systems along with examples are given in detail in [1] and [6].

There are also cases for which communication at all stages is digital and at no stage it is converted into analog. This type of communication is for the devices that are at relatively very short distances i.e. of the order of few meters or even lesser. In the context of this thesis, by digital communication, it is meant that two communicating devices are at large distances i.e. typically 10 meters or more.

There is various numbers of stages that combine together for making digital communications possible. Nice overview about these stages has been presented in [6]. Each of the stage is used in both transmission and reception of data and they are connected in cascading. Moreover, stages are used in reverse in case of receiver as compared to transmitter. Figure 2.1 shows various stages of digital communication system.

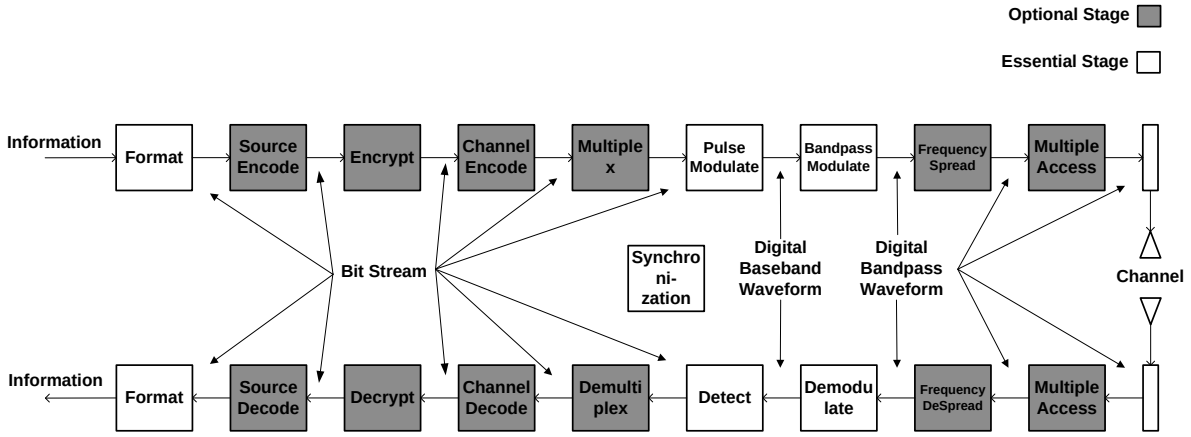

Figure 2.1 Digital Communication System [6]

Figure 2.1 shows an overview of the essential entities required. Many of them are optional and there use is subjected to requirement. Synchronization between receiver and transmitter is also very essential to the overall communication and methods of synchronization are the most vital part that decides the overall performance benchmark of digital receiver in communication system as proved in [7]. All of the relevant concepts that are in the scope of research are given below.

2.2.3.1. Modulation

Modulation is the process of embedding an information containing signal into a second signal so that the transmission of the signal gets easier [8]. Mathematically, it is defined in eq. 2.1.

$$\boldsymbol{y(t) = x(t) * \cos(w_c t + \theta_c)} \quad \text{----------} \quad \textbf{(2.1)}$$

The information bearing signal $x(t)$ is generally known as baseband signal or Modulating signal while the signal which is used to facilitate transmission is non as carrier signal $\cos(w_c t + \theta_c)$ [1]. Figure 2.2 and Figure 2.3 shows baseband and carrier signal respectively.

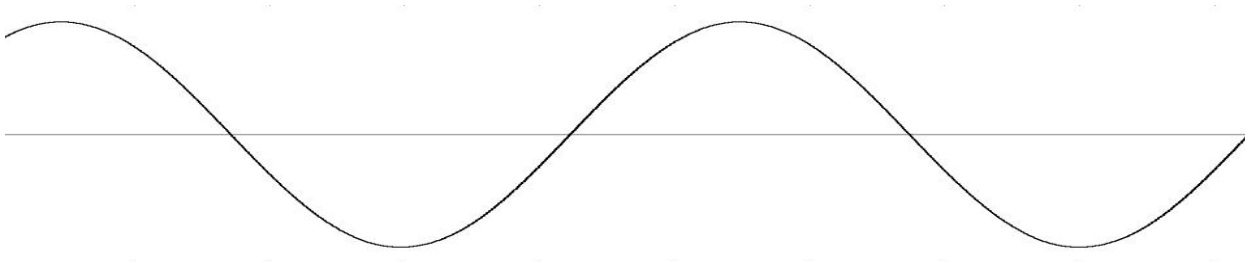

Figure 2.2 Baseband (Modulating) Signal

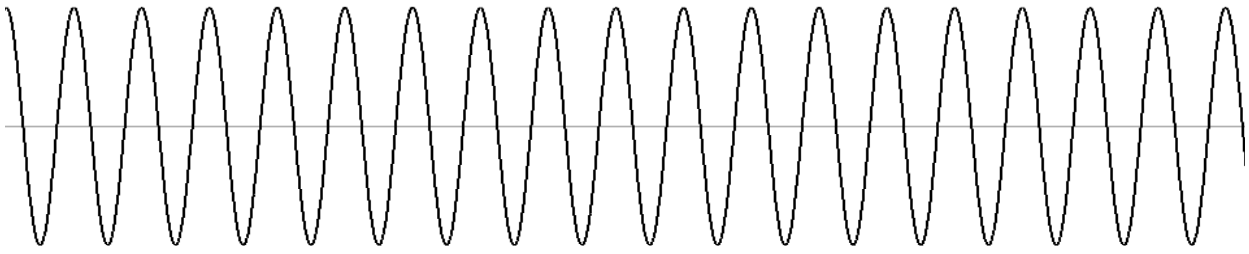

Figure 2.3 Carrier Signal

Widely used types of Modulation are Amplitude Modulation and Frequency Modulation. In Amplitude Modulation, the amplitude of the carrier signal is varied according to the amplitude of the baseband. Figure 2.4 and Figure 2.5 shows modulation of a baseband signal. In frequency

modulation, frequency of carrier is changed on the basis of the amplitude of baseband signal. Both of these types have their own advantages and disadvantages and their use is dependent upon the target application.

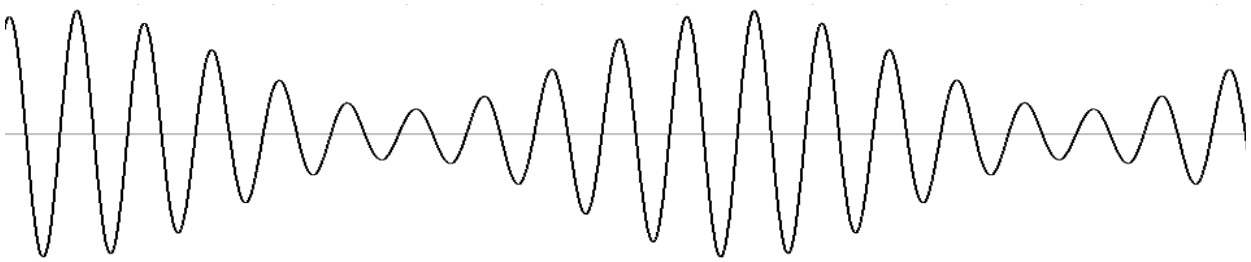

Figure 2.4 Amplitude Modulation

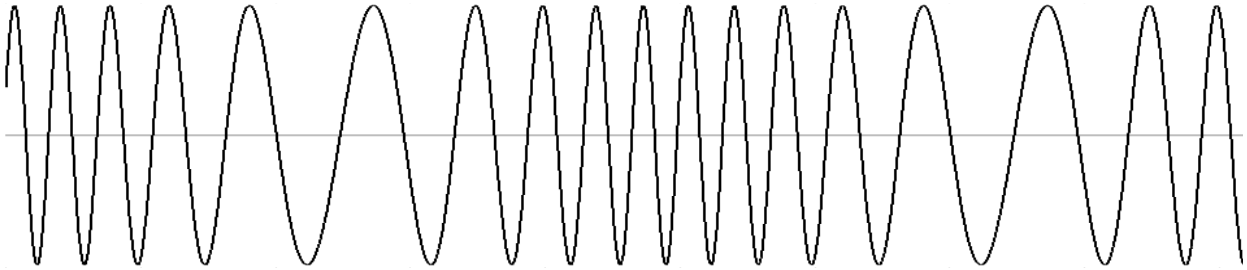

Figure 2.5 Frequency Modulation

If the baseband signal is in digital form, the modulation is replaced by the word shift keying. That is for digital domain, types would be Amplitude shift keying and Frequency shift keying. In this thesis, modulation scheme has been used is PSK (Phase Shift Keying). In this scheme, information of baseband is in the phase of the carrier signal. There can be two out of phase carrier signals each can be used to represent a digital symbol. That is transmitting in phase sinusoidal for zero and out of phase sinusoidal for one. This is known as binary phase shift keying (BPSK) as shown in Figure 2.6.

Similarly, if four sinusoidal waveforms, each out of phase by 90 degrees, are used to represent four symbols of transmission, this type is Quadrature or Quaternary Phase Shift Keying [9]. So, we can generalize that if M numbers of waveforms (separated by (2pi/M) radians) are used each representing a single digital symbol of transmission, then this type of modulation is known as M-ary Phase Shift Keying. Increasing M starts to increase the data rate too but for the same error rate requirement, overall power required to transmit a single symbol also increases [6].

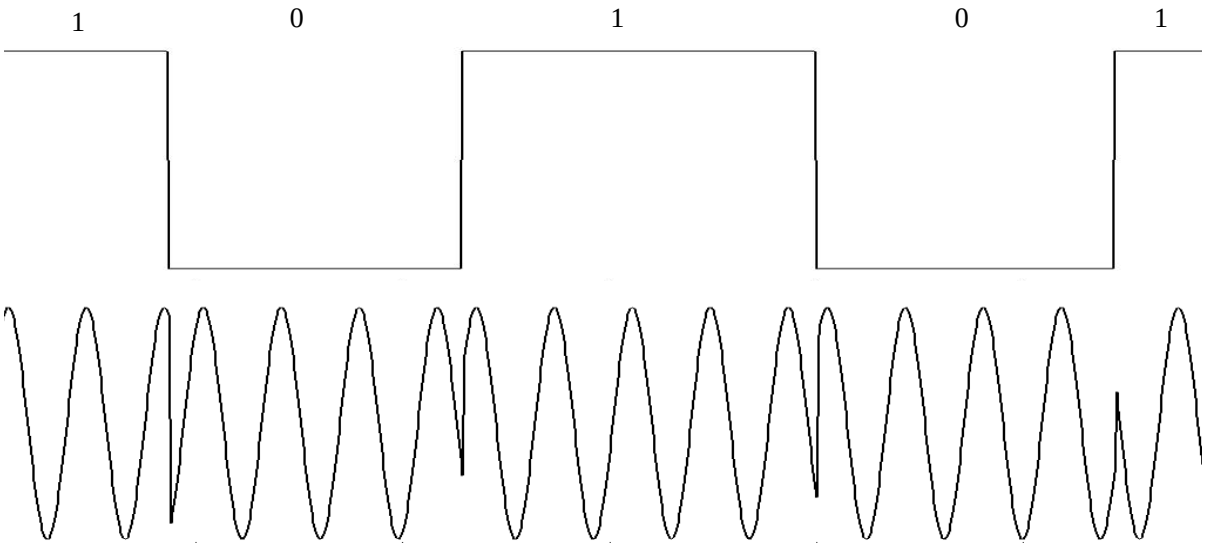

Figure 2.6 Binary Phase Shift Keying

For CDMA modulation, actual data stream is spreaded over the entire allocated bandwidth. It is done by mixing data with pseudo random sequence of much higher data rate. Receiver must know the pseudo random sequence in order to demodulate the received signal. Figure 2.7 shows CDMA modulation.

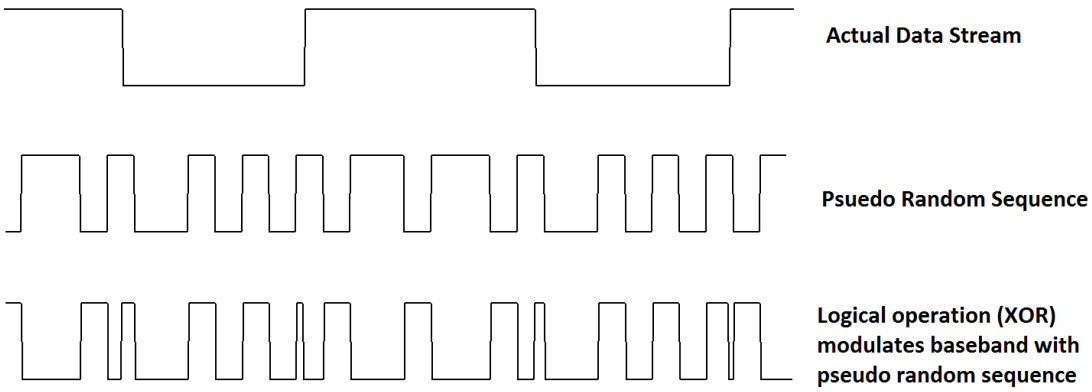

Figure 2.7 CDMA Modulation

2.2.3.2. Demodulation

The process of recovering an information containing signal from modulated signal is known as demodulation [1]. Generally, there are two major demodulation techniques, each with their own pros and cons. These are synchronous and asynchronous demodulation [8].

Synchronous demodulation is the straight forward technique in which baseband signal is recovered by multiplying the regenerated carrier signal, at the receiver, of exactly same frequency as in modulated signal. It is relatively easier technique to execute but regenerating a signal with exact carrier frequency is very difficult to achieve and increases the overall cost of the system. Moreover, with ageing of the components in the circuitry, the overall frequency of regenerated carrier also starts to change. To counter this problem, frequency estimation technique is used. This technique can be realized by including a training signal in transmission [7]. This training signal (already known at the receiver end) is used to estimate the actual frequency of the transmitted signal. After the estimation is made, the signal of that frequency is regenerated at the receiver end.

In case of asynchronous demodulation the need for synchronizing transmitter and receiver is eliminated (as required for synchronous demodulation) [8]. It includes the way of detecting the envelope of carrier signal (used for demodulation of Amplitude Modulation [1]).

2.2.3.3. Signal Energy

There are various ways of measuring the strength of signal. One of them is Signal's Energy. Signal Energy is defined as the area under the square of the signal. Mathematically, it is defined in eq. 2.2. [1],

$$E_x = \int_{-\infty}^{\infty} |x^2(t)|\, dt \quad \text{---------- (2.2)}$$

It is not to be confused with the term known as Energy signal. Energy signal is the signal whose energy is finite ($0 < E < \infty$) [10].

2.2.3.4. Signal Power

This measure of signal tells the rate of energy a signal has. For a signal to be meaning full, it is necessary that its power is less than infinity. Mathematically signal power is defined in eq. 2.3.

$$P_x = \frac{1}{T}\int_{\frac{-T}{2}}^{\frac{T}{2}} |x^2(t)|\, dt \quad \text{---------- (2.3)}$$

It is not to be confused with Power Signal. Power Signal is the one whose energy is infinite but power is finite. [10].

2.2.3.5. Bandwidth

The range of frequencies over which power (or energy) density spectrum is concentrated is known as bandwidth of a signal [10]. Strictly speaking, all of the frequencies present inside a signal form its bandwidth. One of the most fundamental concepts of Signal Processing is that if signal is time limited then it can't be made limited in frequency domain. Vice versa of this is also true [1]. Practically, all signals are time limited and hence their bandwidth is infinite. But most of energy (or power) of that signal is concentrated in specific range of frequencies. So, generally accepted definition of bandwidth is up to range of frequencies that make certain proportion of energy (or power) of signal. This proportion may be 75% or 90% or 95% or 99% depending upon the scenario [10].

If bandwidth of signal is concentrated about zero, it is known as low-frequency signal. Similarly, if bandwidth is concentrated on high frequency signals, the signal is of high frequency type. If it contains the hybrid sort of signal that contains the mix up of low and high frequencies, type of signal is known as band pass signal [10]. Figure 2.8 shows each of the signal type.

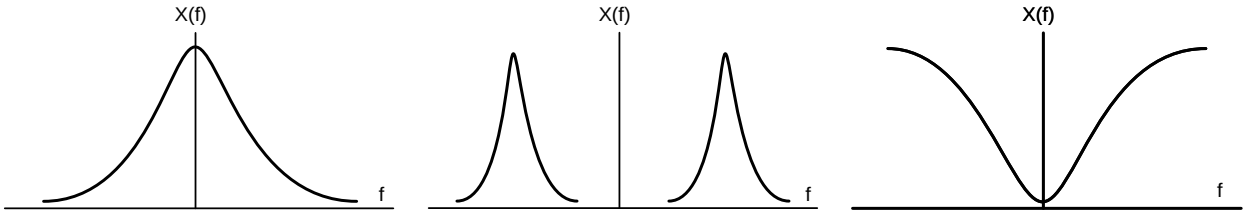

Figure 2.8 (a) Low Frequency Signal (b) Medium (Band pass) Frequency Signal (c) High Frequency Signal

Bandwidth is an extremely precious resource so it is handled very carefully. All of countries in the world have their respective regulations regarding the use of bandwidth. Its unauthorized use

is considered as crime. The reason being the fact that if more than one user is transmitting signal at the same place and time in same bandwidth, interference will occur which will cause the receiver fail to communicate.

2.2.3.6. Signal to Noise Ratio (SNR)

It is the metric of measuring the overall quality of a signal. As a signal propagates through any channel, it gets affected by the noise in the channel. Effect of noise is more as the distance from transmitter is increased which in turn causes to increase the overall error rate at the receiver end. The effect caused by this noise cannot be reversed. This noise is known as thermal noise and it corrupts the signal in an additive fashion. One of the biggest advantages of digital communication is the graceful degradation in noisy environment [6]. Mathematical representation of SNR is given in eq. 2.4.

$$\text{SNR} = \frac{\text{Signal Power}}{\text{Noise Power}} \quad \text{---------- (2.4)}$$

2.2.3.7. Intersymbol Interference (ISI)

In the field of digital communications, filters are of prime importance. They are frequently used throughout in the different stages. One of the most important stages is before transmitting the signal in both wired and wireless transmission. Signal has to be made band limited in this stage. While making the signal band limited or by suppressing those frequency contents that are not allowed due to bandwidth constraints, the signal shape distorts in time domain [1]. Figure 2.9 illustrates how symbols get smeared and interfere with each other after filtering.

Moreover, the channel through which signal will propagate also contain some undesirable properties. It includes additive noise, multipath effects and for wired channel, the baseband signal faces non-uniform reactance [6]. Channel noise adds amplitude errors and frequency response of a channel causes further smearing of transmitted symbol [8].

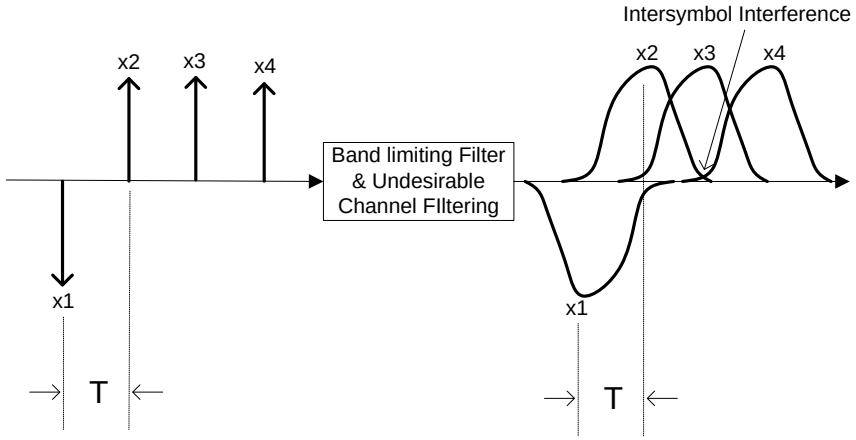

Figure 2.9 Effects of limiting bandwidth of baseband signal before transmission

To avoid the intersymbol interference, the pulse is shaped in a way that even after making it band limited, intersymbol interference does not occur. This technique is known as pulse shaping [1] [6] [8]. The idea behind this technique is to remove the interference at the time instant at which the pulse is required to be detected [1]. Nyquist proposed different criteria of shaping the pulse that could reduce or even nullify the intersymbol interference.

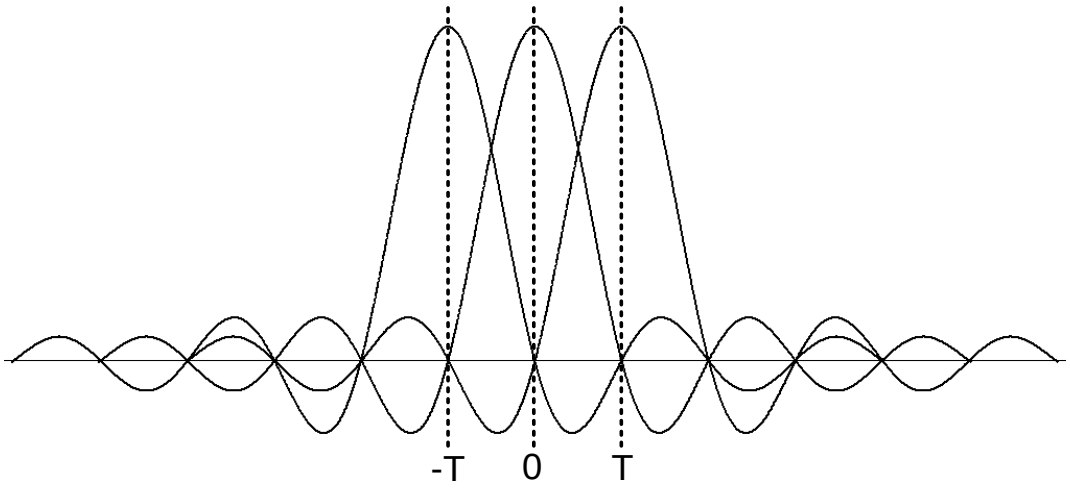

Figure 2.10 Minimum Bandwidth Pulse that also satisfies Nyquist zero intersymbol interference criterion

One of the most famous ideas of Nyquist about eliminating ISI is given in [11]. He stated that, if the pulse is shaped such that it is zero at the sampling time of other symbols, ISI can be avoided. Note that, the value of interfering symbol will only be zero exactly when symbol value is meant

to be sampled and not before or after the sampling time. By following this criterion, we can make sure that even after band limiting a signal, ISI will be avoided. The pulse in Figure 2.10 is the minimum bandwidth pulse that satisfies the Nyquist criterion of zero ISI [1].

2.2.3.8. Multiple Channel Access:

In case of two or more wireless radios operating within range and trying to talk to some other radio, there is a chance of collision. To avoid this, it is quite pertinent to use some anti-collision methods. The biggest problem while using these methods is the way multiple signals are handled together. Careless handling may cause the signal quality to deteriorate because of the interference from other signals [12]. Anti-collision methods include:

Space Division Multiple Access (SDMA)

In this kind, same frequency transmitters are separated in space. They can transmit their data where they are at enough distance from other same-frequency transmitter such that their transmission cannot be collided. Simply, each same frequency transmitter has its own exclusive territory.

Time Division Multiple Access (TDMA)

In this type, transmission from different transmitters is divided in time. In simpler words, when one transmitter transmits, all other same frequency transmitters in its proximity are banned from transmitting a signal. This phenomenon also occurs when there are multiple devices and only one transmitter, the system requires TDM type in which transmitter is multiplexed and sends data of every device for some predetermined amount of time.

Frequency Division Multiple Access (FDMA)

Multiple devices are allowed to transmit within their pre allocated frequency ranges. Using this Phenomenon, multiple devices can transmit at same time and at same location.

Code Division Multiple Access (CDMA)

Code division multiplexing access is one of the multiple channel access methods in which multiple transmitters transmit data at same location, time and frequency. The transmission of all transmitters is spreaded in entire available frequency spectrum. The key difference between their transmissions is the spreading code or device specific transmission law. [12]. This transmission

specific code is known to the receiver in advance. Receiver uses the method of co relation for detection of the transmission of interest.

Wideband Code Division Multiple Access (WCDMA):

This is the variant of CDMA in which upload and downloaded streams are divided into time or frequency slots. It contains the integration of multiple channel access methods. For frequency division/ Code Division multiple access, uplink and downlink have different frequencies. For time division/Code division multiple access, uplink and downlink are separated by time slots [12]. All characteristics contained by CDMA are also the part of WCDMA.

2.2.3.9. Spread Spectrum Techniques:

In CDMA, signal can be spreaded using different techniques, widely used are following.

Direct Sequence spread spectrum (DSSS)

In this type of spreading, the modulation of carrier is done for the second time with some pseudo random code. This code differs from the information in the sense that its bit rate is higher and usually it does not effectively contain any information. It is also known as ‘chips’ and the bit rate is known as ‘chip rate’ [13]. High bit rate of code causes the information containing carrier to wide spread in frequency which has multiple advantages. The information in the signal is decodable only when the chip sequence is known a priori to the receiver. This causes the resistance against unauthorized listening of the signal. Figure 2.11 shows the schematic of DSSS.

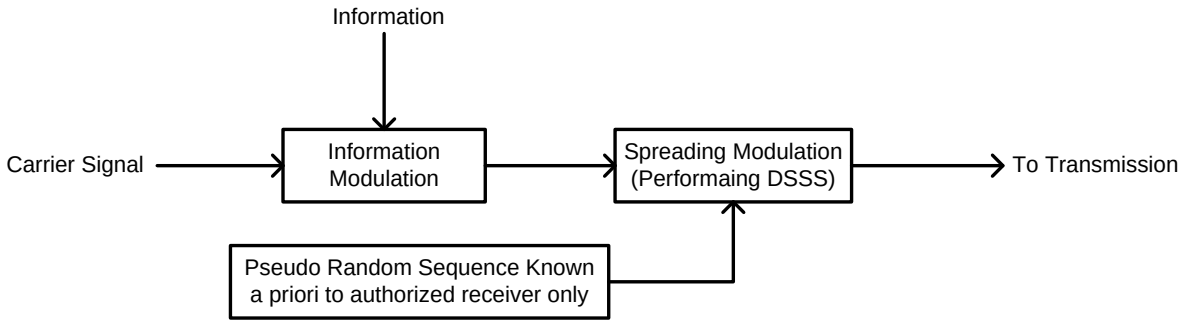

Figure 2.11 Direct Sequence Spread Spectrum [13]

Moreover, multiple transmitters with different chip sequence can transmit the signal at same frequency, time and space which decreases the bandwidth requirement. Also, being spreaded at wide frequency range, it is less vulnerable to jamming due to the fact that jammer has to transmit a confusing signal to extremely wide range of frequency to be effective which will require huge amount of power.

Frequency Hopping Spread Spectrum (FHSS)

In this type of spreading, the carrier containing information is hopped pseudo randomly over different frequency within permitted bandwidth [14]. For different systems, the speed with which carrier frequency hops over different frequencies is different. This pseudo random hopping sequence of carrier frequency is known a priori to the receiver.

2.2.4. Digital System Design

When it comes to mapping DSP systems on fully customizable target technology i.e. Field Programmable Gate Array, it is quite important to know that fully customizable designing capability acts as both advantage and disadvantage. Advantage is due to the fact that designer can design the system by purely using his own creativity effectively independent of any architecture designed before which gives him full control to create perfectly optimized system for given set of requirements. And disadvantage is that, every new part of design has to be fully tested and this testing procedure requires extra effort. Moreover, if carelessly created, it becomes difficult to change or to add new features in the system.

As soon as FPGAs emerged into the market, they have become ideal for implementing real time DSP systems. FPGA designer companies designed internal structure of FPGAs in a way that makes it ideal for doing multiplication and accumulation at extremely fast rate; a quite common operation in DSP algorithms. Over the period of its development, the system designer's capability and confidence increased due to new advances in this technology allowing them to design a whole system on a single chip (SOC). Market leaders Xilinx and Altera both are getting confidence from their customer organizations by making FPGA system designing tools that are easier to learn and can implement high caliber digital systems in lesser time.

The development environment used for programming depends on the company that has designed the FPGA chip used within. Development board is the Printed Circuit Board (PCB) that contains the chip of FPGA physically connected with different number of peripherals. Xilinx made boards include Spartan 3, Spartan 3A, Vertex Series, Spartan 3A-DSP etc. and Altera's FPGA leading board designer is Terasic Technologies who has made series of DE-2 and DE-3 boards.

For Xilinx FPGAs development environment is Xilinx ISE in which a project can be created and navigated, although it provides basic ability to write a program in prescribed language with some facilities but other softwares designed by Xilinx can also be used to refine the design. Each of these softwares is specialized for a specific purpose. Some of these softwares are Xilinx PlanAhead, Chip Scope, Embedded Development Kit, System generator, iMPACT and Core generator etc. Detailed information on these softwares can be obtained from manufacturer's website. For Altera FPGAs, Quartus 2 designed by Altera Corporation is used.

2.2.4.1. Suitable Programming Languages

The most basic languages used for its programming is Verilog and VHSIC Hardware Description Language (VHDL). The basic purpose of this language is to give the description of hardware needed to be built. The major difference between this language and other commonly used languages such as C, C++, C# etc. is that the description given by this language is constructed in hardware and its lines of code generally works in parallel. Verilog has some of its constructs similar to C Language. Moreover, C language is also used quite extensively in the design of IP cores in FPGAs. The role of C language is to program the hardcore or soft core predesigned readily available processors so that those processors can work in conjunction with the custom designed hardware. For simulation of purposes, stimulus can also be written in Verilog.

2.2.4.2. Combinational and Sequential Circuits

In any digital design system, there are two types of circuits, combinational and sequential. Combinational circuits does not contain any memory elements, moreover, they also have no feedback input. Moreover, they also do not synchronized on clock ticks. On the other hand sequential circuits can produce delays in their outputs due to presence of memory elements in them and can have feedback input.

2.2.4.3. Floating & Fixed Point Format

To convert computation of DSP algorithms into live implementation, the numbers are to be mapped on digital voltage lines for arithmetic operations. There are two types of floating point numbers; single and double precision floating point. Processors having floating point arithmetic capability are known as floating point processors. The detail of floating point format and its arithmetic is given in IEEE standard for floating point arithmetic.

Processors that have floating point number processing capability are bound to do three operations automatically: exponent adjustment, mathematical operation and normalization. Floating point numbers are used where any number is required to be represented over extremely large dynamic range with high accuracy. In DSP algorithms, this is generally not a requirement. Implementation of floating point arithmetic is more power and area hungry while giving lesser performance, hence it is generally avoided [5].

Fixed point implementation has lesser area requirements and is ideal for operation in power constraint embedded systems. However, due to its less ability to perform automatic operations, this format is required to be traced and guaranteed by the designer of the system. In case of over flow or under flow, the handling is quite different from floating point numbers and if handled carelessly, it may lead to incorrect results. Shortly the track of fixed point format is required to be done manually after each set of operation. Moreover, it is flexible in the sense that, depending upon the designer, custom number of bits can be assigned to fractional and integral part which gives extended control over the accuracy and range of number to be represented.

Representation of fixed point format is known as Q point format. For example Qn.m represents a number of n + m bits with n and m bits are used to represent integral and fractional part of the number respectively. If two numbers with Q4.5 and Q6.4 are added, result will be in the format of Q6.5 (largest integral and fractional number is selected). In case of multiplication result will be dependent upon the sign of each number and results will be as shown in Table 2.1.

First Number	Second Number	Result
Unsigned $Qn_1.m_1$	Unsigned $Qn_2.m_2$	Unsigned Q $(n_1 + n_2).(m_1 + m_2)$
Signed $Qn_1.m_1$	Unsigned $Qn_2.m_2$	Signed Q $(n_1 + n_2).(m_1 + m_2)$
Unsigned $Qn_1.m_1$	Signed $Qn_2.m_2$	Signed Q $(n_1 + n_2).(m_1 + m_2)$
Signed $Qn_1.m_1$	Signed $Qn_2.m_2$	Signed Q $(n_1 + n_2 - 1).(m_1 + m_2 + 1)$

Table 2.1 Cases of Fixed Point Number multiplication [5]

To counter this problem, hardware embedded specially designed clock lines that supply clock ticks in different blocks are present inside FPGA. Moreover, care is also required to be taken while working with clock as it is not a good practice to divide a clock using cascade of counters and then giving this clock to some other module to operate. In order to perform operations on the clock, Digital Clock Manager (DCM) blocks are used [15].

2.2.4.4. Finite State machines

Finite State machines (FSM) are ideal for those systems that require sequence of operations to fulfill the desired task. It has the ability to make logical decisions on the basis of previous operation or its output values. Each state is a stable entity which system can occupy. The transition from one state to another state is done under the control of outside world-input [16]. One typical design may include FSM that can generate control signals specific for each state. The general design of FSM includes state register which represents the current state of the machine. The most important factor in this regard is the encoding of different states within the state register.

There are different encoding schemes for state register as binary encoding, one hot encoding, almost one hot encoding and gray encoding. Each of these schemes have their advantages and disadvantages and are used depending upon the requirement of the system. For example for one hot encoding, area requirement is high but transition between different states is fastest due to state transition logic is simplest. Hence it is highly suitable for performance hungry systems. Similarly, the binary encoding is area efficient but its transition logic is complex which consumes more time in state transition [5].

State machines are based on three major entities; next logic decoder, clock based memory units and output logic decoder. Historically, there are two distinct types of state machines in which all of these entities are used in different manner. State machines can be synchronous if they get input generally on the edge of clock (positive or negative) and are asynchronous if their input decoding mechanism is of combinational type.

Mealy Machine

It is the type of FSM in which both output and next logic decoding depends upon present state and current user inputs to the FSM. Figure 2.12 illustrates Mealy Machine.

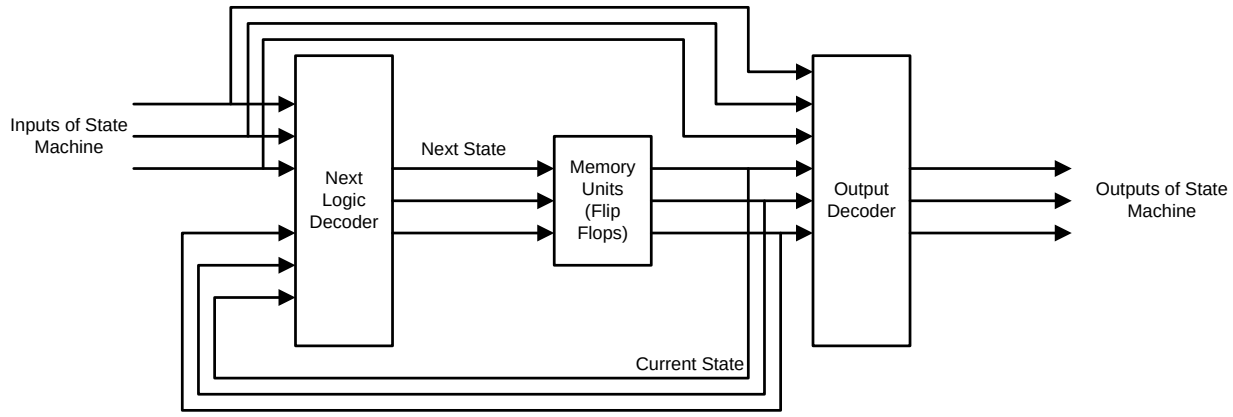

Figure 2.12 Mealy Finite State Machine Architecture [16]

Moore Machine

In contrast to the former type, Moore Machine is the one whose output is decoded only on the basis of current state. Figure 2.13 shows the architecture of Moore Machine.

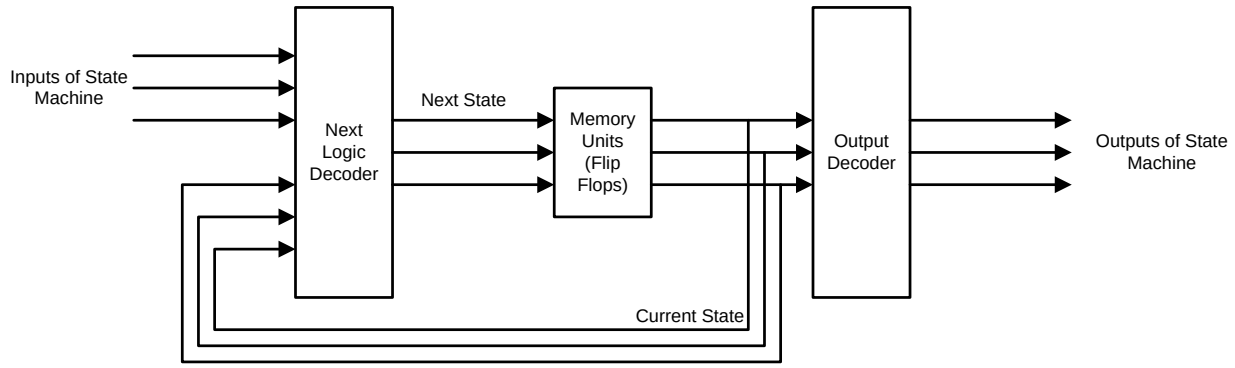

Figure 2.13 Moore Finite State Machine Architecture [16]

2.2.4.5. Latch & Flip Flop

Latch and Flip Flop both act as a memory unit in circuitry. But latch is clock less while flip flop operates on clock. There are several design approaches that may lead to the formation of latches inside the design. FPGA vendors generally recommend designers to avoid the latches as they may lead the overall design to malfunction or become technology dependent. Latches can be inefficient and cause complications in timing analysis [17]. Care is to be taken while writing the code. Following approaches should be followed when writing the code in Verilog.

- For each case statement, always use default case.
- Define the state of all relevant registers for every case in all case statements, even in default case too.
- Define the states of all relevant registers in both if and else statements.
- For synchronous always block, use non-blocking assignment operator for assigning values to registers.
- For combinational always block, use blocking assignment operator for assigning values to registers.

2.2.4.6. Block and Distributed Memory Units

Several algorithms require block of data to be saved temporarily somewhere in the system and then to be used later on for further computation. Moreover, high speed data read/write from the temporary storage is also a requirement. To fulfill this, on chip block RAM are provided on

FPGA. Generally, block RAM read/write both take one clock cycle and these blocks are well placed such that they are accessible from all parts of FPGAs. Typical size of block RAM is several kilobits and it depends on the type of FPGA used.

Distributed RAM is a Look Up Table (LUT) present in every Configurable Logic Block (CLB) of FPGA. Distributed RAM is faster than Block RAM but it is very small in size. Distributed RAM is faster than block RAM but it is small in size. Moreover, in Xilinx FPGAs, distributed RAM is of both single and dual port [15].

If memory requirement is high for complex systems then either off chip RAM is used or multiple FPGAs are connected and used together. Off chip memory has high capacity but their speed is much lower than that of on chip RAM (both distributed and block RAM). Off chip RAM and multiple FPGAs working together is generally subjected to the design of the development board of FPGA.

2.3 Generic WCDMA Receiver Abstract Design

Based upon the general characteristics, in which receiver must be able to recover the baseband signal without synchronous demodulation, there are multiple choices for the design. It includes the signal processing immediately after the Analog Front End (AFE) from detecting the start of burst to the actual stream of information intended for communication. The detailed abstract diagram of complete SDR receiver is shown in Figure 2.14.

As already discussed, this receiver is the part of SDR project initiated by the CARE organization. There are series of algorithms that need to be performed on the signal to get the required output. It is quite important to mention here that the order with which they are performed on the signal is the ease and availability of all factors required for their operation.

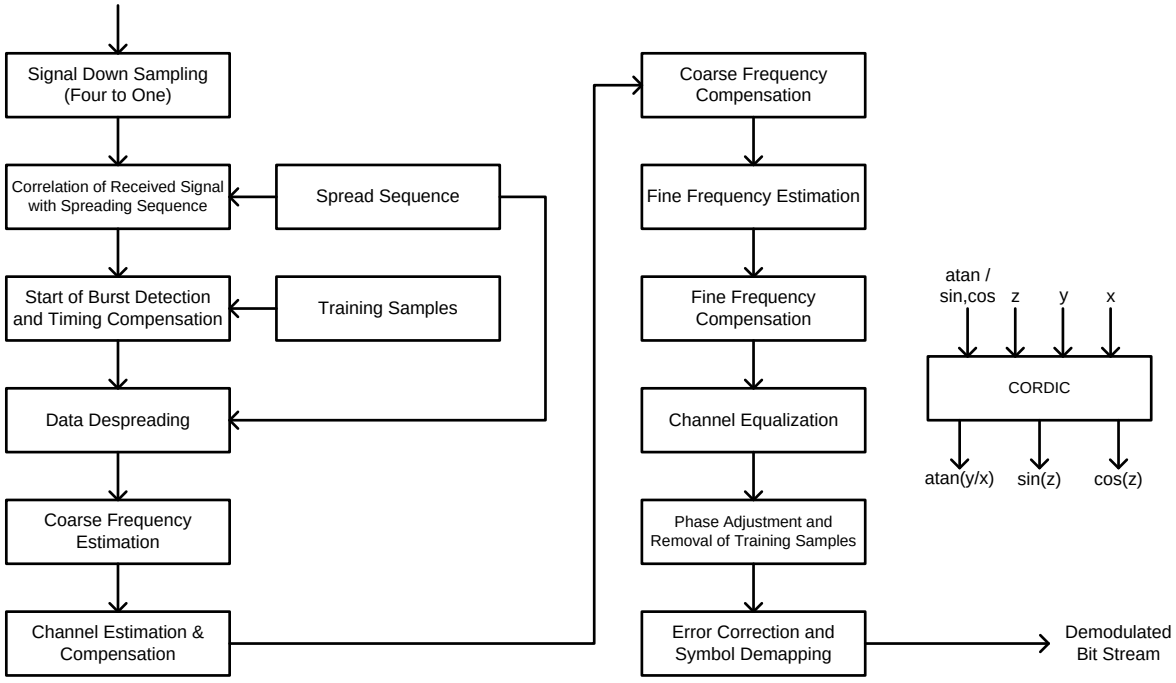

Figure 2.14 Abstract Diagram of Receiver

2.4 Throughput Calculation of Current Implementation

This section discusses some requirements and specifications of the design which are preset before the implementation of necessary parts SDR receiver.

2.4.3. System Specifications

This section tells about the specifications of the SDR receiver to be designed. Table 2.2 shows the parameters of the receiver to be designed.

Serial Number	Parameter	Design Value
1	Training Length	32 (64 for future systems)
2	Spreading Factor	16
3	Data Length	288
4	Modulation Index	4
5	Modulation Schemes	QPSK
6	Target throughput	384 kbps
7	Forward Error Correction	1/2
8	Chip Rate	6.83 Mcps
9	Up sampling Factor	4

Table 2.2 System Specifications

2.4.4. Throughput Calculation

At each part of the receiver, the throughput requirement may be scaled up or down depending upon the nature of operation it is doing on the data stream. Chip rate has been calculated in eq. 2.5.

$$\text{Chip Rate} = \frac{\text{Throughput} \times \text{Spreading Gain} \times (\text{Data length} + \text{training length})}{\text{Bits per Symbol} \times \text{Forward Error Correction} \times \text{Data Length}}$$

$$= \frac{384000 \times 16 \times (288 + 32)}{2 \times 0.5 \times 288} = 6.83 \text{ Mcps} \quad \text{----------} \ (2.5)$$

Actual Bandwidth in digital domain is after up sampling the chip rate by 4. It can be calculated as,

$$\text{Actual Bandwidth After Upsampling} = \text{Up sampling Factor} \times \text{Chip Rate}$$

$$= 6.83 \text{ Mcps} \times 4 = 27.307 \text{ Mcps} \quad \text{----------} \ (2.6)$$

Using the calculation shown above, the throughput require for each constituent component can be easily calculated. It is shown by Figure 2.15.

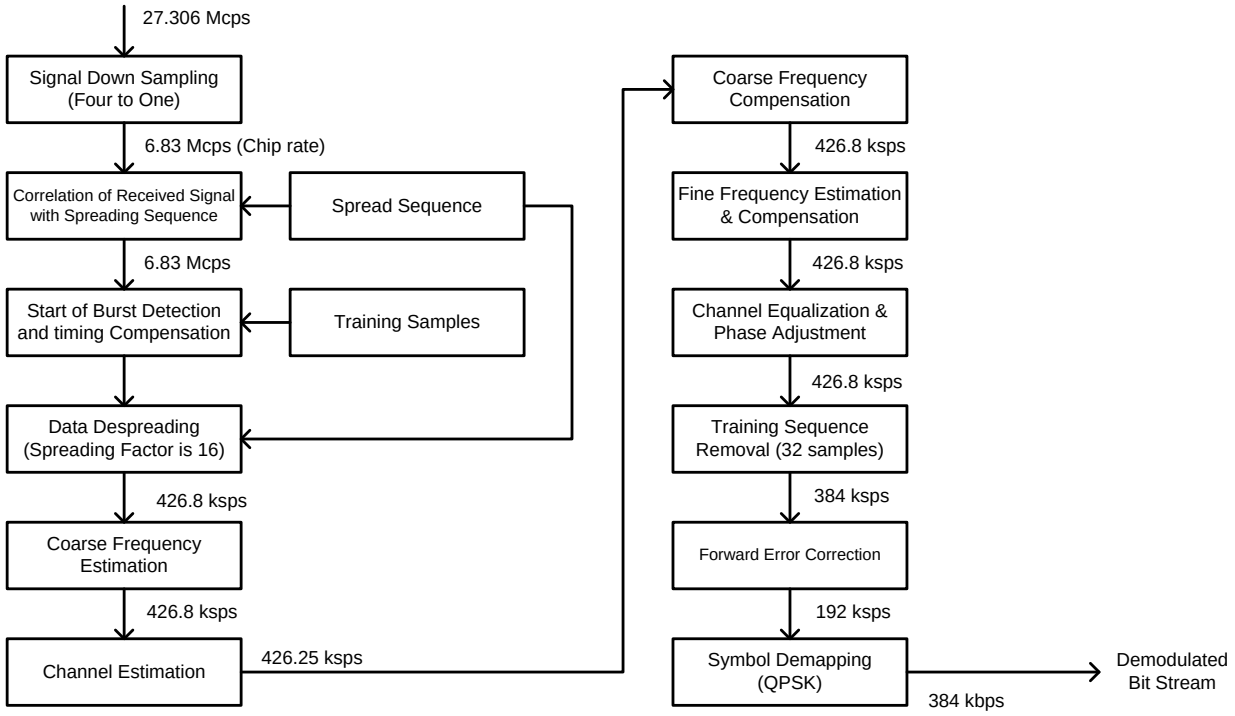

Figure 2.15 Component wise throughput Calculation

2.4.5. Doppler Shift Calculation

.The Doppler shift for the system can be calculated as shown in eq. 2.7.

Maximum Relative Speed $\approx 30 \sim 60$ km/h

Speed of Light $= 3 \times 10^8$ m/s

Carrier Frequency $= 233$ MHz

Intermediate Frequency $= 21.4$ MHz

$$\text{Doppler Shift} \approx \frac{\text{Maximum Relative Speed}\left(\text{in}\,\frac{\text{m}}{\text{s}}\right) \times \text{Carrier Frequency}}{\text{Speed of Light}}$$

$$\approx \frac{(30 \sim 60) \times 233 \times 10^6}{3 \times 10^8 \times 3.6} \approx 6.47 \sim 12.94 \text{ Hz} \text{ - - - - - - - - - - - - (2.7)}$$

2.5 Resources Available for Implementation

The target FPGA for implementation of various parts of SDR is xc3sd3400a-4cs484. This device is from the family of Spartan 3-A DSP FPGAs from Xilinx. It has quite large number of resources especially DSP blocks which are highly required in computationally complex implementations. The resources of this device are shown in the table below.

Resource	Quantity
Total Slices	23,872
Total 4 input LUTs	47,744
Total Slice Flip Flops	47,744
I/O Pins	309
DSP Blocks	126
Block RAM	126

Table 2.3 Total Available Resources in xc3sd3400a-4cs484

Algorithms up to data despreading are already implemented. The resources consumed by previous implementations and actual available resources for systems to be implemented are shown in the table below.

Resource	Consumed Resources by already implemented system	Free Resources	Percentage of Free Resources
Total Slices	9,891	13,981	59 %
Total 4 input LUTs	13,379	34,365	72 %
Total Slice Flip Flops	9,621	38,123	80 %
I/O Pins	96	213	69 %
DSP Blocks	69	57	46 %
Block RAM	22	104	83 %

Table 2.4 Actually Available Resources for Current Implementation

2.6 Relevant Algorithms and Implementations

This section discusses the relevant algorithms, implementations and widely practiced designs on the relevant topics.

2.6.3. CORDIC (Coordinate Rotation Digital Computer)

One of the very important components of the SDR is CORDIC module. This algorithm was introduced by J. E. Volder in 1959 to fulfill the real time airborne computation [18]. Initially it

was targeted for military equipment only but as time passed, it started to be used in application for commercial purposes too. It is one of the mainstream algorithms for trigonometric functions. This was presented as a unified algorithm by J.S. Walther [19]. One of the inherent properties of this algorithm is it does valid calculations for only limited set of inputs. Researchers have solved this convergence problem mostly by using trigonometric identities.

It is basically an iterative algorithm that can be used in two modes. One mode is rotation and other is vectoring mode. There are three basic equations, eq. 2.8 – eq. 2.10, that operate under these modes iteratively.

$$x_{i+1} = x_i - \sigma_i 2^{-i} y_i \quad \text{---------- (2.8)}$$

$$y_{i+1} = y_i + \sigma_i 2^{-i} x_i \quad \text{---------- (2.9)}$$

$$z_{i+1} = z_i - \sigma_i \tan^{-1} 2^{-i} \quad \text{---------- (2.10)}$$

where $i = 1$ to (Total Iterations $- 1$) and σ_i can be -1 or 1

	Initialization	**Final Values (after N Iterations)**
Rotation Mode $\sigma_i = \begin{cases} 1 \; if \; z_i \geq 0 \\ -1 \; if \; z_i < 0 \end{cases}$	$x_0 = \frac{1}{k}$ $y_0 = 0$ $z_0 = \theta$ $where \; k = \prod_{i=0}^{N-1} \sqrt{1 + 2^{-2n}}$	$x_n = \cos\theta$ $y_n = \sin\theta$ $z_n = 0$
Vectoring Mode $\sigma_i = \begin{cases} -1 \; if \; y_i \geq 0 \\ 1 \; if \; y_i < 0 \end{cases}$	$x_0 = \frac{x}{k}$ $y_0 = \frac{y}{k}$ $z_0 = \theta$	$x_n = \sqrt{x^2 + y^2}$ $y_n = 0$ $z_n = \theta + tan^{-1}(y/x)$ $where \; k = \prod_{i=0}^{N-1} \sqrt{1 + 2^{-2n}}$

Table 2.5 Functions Calculation using CORDIC

Generally the total number of iterations is 16 or 32. Let Total Number of Iterations to be N. Table 2.5 summarizes the behavior of CORDIC Algorithm. It includes only circular computational behavior because only this part is used in the implementation of SDR. Linear and hyperbolic computations of CORDIC are given in detail in [20].

In rotation mode, vector is rotated recursively using known rotations. After N number of known rotations, vector reach at the desired angle and its x and y components are $\cos\theta$ and $\sin\theta$ respectively. In vectoring mode, basic concept is to have initial coordinates of the vector whose angle is required to be calculated. It presumes that initial x coordinate is always positive and initial y coordinate is any arbitrary number. CORDIC algorithm iterates the y coordinate such that it becomes zero and net calculation is done which results the angle through which the vector is rotated. This calculation gives $tan^{-1}(y/x)$. Figure 2.16 illustrates single iteration.

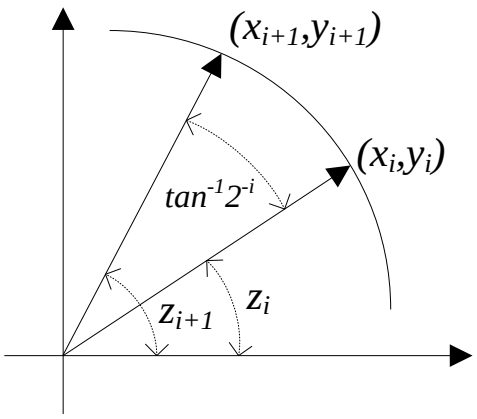

Figure 2.16 Single Iteration of CORDIC Algorithm

This algorithm has been implemented in countless number of applications. Generally, the proposed design and implementations are requirement specific. One of the most comprehensive researches is done in [21]. It includes the detailed literature about the capabilities of CORDIC algorithm. This paper also discusses different types of architecture each targeted to different set of requirements for example iterative type, bit serial type, fully unrolled architecture and hybrid types. One of the Hybrid type architecture is derived from fully unrolled type in which all additions and subtractions are done in bit serial form. This paper also includes the details of one of the first CORDIC implementation in FPGA.

2.6.3.1. Previous Implementations

While considering the CORDIC algorithm for hardware implementation, results are always based on approximation. These errors are introduced due to the quantization. However, by increasing the resources, designers can reduce errors as per their requirement. A detailed analysis on quantization errors in CORDIC have been conducted in [22] for both floating and fixed point hardware implementations. It describes and presents the mathematical theorem for various scenarios like error propagation in CORDIC iterations and rounding errors in normalization operation.

With the target implementation of SDR, [23] has implemented a pipelined based architecture that calculates the value of sine and cosine. They have implemented an angle sequencer that is used to generate sine/cosine waves. This sequencer will generate saw tooth wave which will be given as an input to the CORDIC system.

Neji and Boudabous [24] have implemented a CORDIC algorithm that is targeted to fingerprinting algorithm. Their design precision is up to 14 bits but the design which they have proposed has convergence problem. The architecture proposed in [23] also has the convergence problem. The average error for their exponential and tangential designs is 0.005% and 0.01% respectively.

Some of the generic and fast methods are proposed in [25] and [26]. Authors of [25] have proposed an architecture that is known as branching CORDIC method. In this method, two conventional CORDIC iterations are run in parallel. Although this algorithm is fast but it consumes more area than many other designs which is the drawback of it. Few alternative implementations are presented in [26] that results in increase speed and reduced area. One of the proposed modification in conventional design is the use of carry free adders instead of carry propagate adders.

Other researches include scalable pipelined architecture has been designed and implemented in [27] in which they claim to have memory less architecture. Number of iterations for the computation is 14 bit and overall architecture is 16 bit. Both [28] and [29] have compared their designs with software implementations. Design in [28] has shown that implementation on certain

family of FPGA is 27500 times faster than that of implemented in software on 2 GHz T7300 processor, hence it is better to implement CORDIC in hardware for satellite altitude determination. Their architecture is of iterative type and based on barrel shifter in which they have limited their implementation to the calculation of sine and cosine functions only.

Similarly in [29], NIOS 2 soft core processor has been used to show the difference in computation time of CORDIC in software and hardware. Architecture is 16 bit pipeline based. Special instruction in NIOS 2 processor has been used to invoke the hardware accelerated circuit of CORDIC and ratio of calculation of the time taken by same instruction in software and hardware is 26. In this way, it is shown that by using the custom hardware designed in FPGA with soft-core processor, one can easily increase the overall processing efficiency.

2.6.4. Coarse Frequency Estimation & Compensation

This algorithm is very convenient when demodulation of the received signal at the receiver end is done without the knowledge of exact frequency of carrier signal. Algorithm selected for implementation does this estimation by using training samples in transmitted data. This training data is already known to the receiver. Using this, the rough estimation of frequency is done. This algorithm is given in [30].

Let us define an array of size N + 1, where N is the total number of samples in training sequence.

$$\mathbf{b(k)} = \frac{\mathbf{1}}{\mathbf{2}}\left(\left(\frac{\mathbf{N^2-1}}{\mathbf{4}}\right) - \mathbf{k(k+1)}\right) \quad \text{----------} \ (2.11)$$

where $k = -\frac{N}{2}, -\frac{N}{2}+1, \dots, \frac{N}{2}-1, \frac{N}{2}$

Received training sequence and already known noiseless training sequence to the receiver are given in eq. 2.12 and eq. 2.13.

$$\mathbf{r(k)} = \mathbf{r(0)}, \mathbf{r(1)}, \mathbf{r(2)}, \mathbf{r(3)}, \dots, \mathbf{r(N-1)} \quad \text{----------} \ (2.12)$$

$$\mathbf{a(k)} = \mathbf{a(0)}, \mathbf{a(1)}, \mathbf{a(2)}, \mathbf{a(3)}, \dots, \mathbf{a(N-1)} \quad \text{----------} \ (2.13)$$

On the basis of described equations, the coarse frequency estimation can be calculated as

$$\hat{v} = \frac{1}{2\pi} arg\left(\sum_{k=0}^{N-2}\left(b\left(-\frac{N}{2}+k+1\right).\left(\frac{a(k+1)}{a(k)^*}\right)^*.r(k+1).r(k)^*\right)\right) \text{ ---------- (2.14)}$$

Once the frequency has been estimated, the received data stream can be coarse frequency compensated using the following equation. Suppose the data as of length '*m*', then compensated data can be represented as,

$$\mathbf{Compensated_{Data(z)}} = \mathbf{Uncompensated_{Data(z)}}.\mathbf{e^{-i2\pi\hat{v}z}} \text{ ---------- (2.15)}$$

where $z = 0$ to $m - 1$

Note that this estimation is quite rough and does not calculate the exact frequency within tolerate able limits. For this purpose, fine frequency estimation algorithm is used.

2.6.4.1. Previous Implementations

Targeted to different set of requirements, different implementations address the problem of coarse frequency estimation. A total of three algorithms about carrier offset estimation as given in eq. 2.16 to eq. 2.18 have been compared in [31].

$$\mathbf{\hat{M}_1} = \mathbf{max_{l\in L}}\left\{\sum_{k=1}^{W}\left|\mathbf{Y_{m+1,k+l}}, \mathbf{Y^*_{m,k+1}}\right|\right\} \text{ ---------- (2.16)}$$

$$\mathbf{\hat{M}_2} = \mathbf{max_{l\in L}}\left\{\left|\sum_{k=1}^{W}\mathbf{Y_{m+1,k+l}}, \mathbf{Y^*_{m,k+1}}\right|\right\} \text{ ---------- (2.17)}$$

$$\mathbf{\hat{M}_3} = \mathbf{max_{l\in L}}\left\{\sum_{k=1}^{W}\left|\mathbf{Y_{m,k}}, \mathbf{X^*_{m,k+s}}\right|\right\} \text{ ---------- (2.18)}$$

W is total number of samples in training sequence whereas L & l is the sliding window of frequency offset estimation. X and Y are the transmitted and received training symbols respectively. The value in the subscript shows the number of actual sample to be processed.

Results show that algorithm in eq. 2.16 works perfectly fine with unsynchronized symbol timing but get badly affected by multiple path effects of channel. Moreover, eq. 2.17 gets badly affected by unsynchronized symbol timing of training sequence but works well in multiple path environments. Algorithm of eq. 2.18 gives the best result of all in which it can deal with multiple path effects and does not get affected by error in symbol timing.

Quick comparisons of performances of multiple coarse frequency estimation algorithms such as L&R [32], M&M [33] and Fitz [34] have been performed in [35]. It also contains the efficient

implementation of L&R algorithm in FPGA with targeted application of Digital Video broadcasting via satellite communication. The architecture presented in this paper is of shared nature. There architecture is of three stages; first one is the buffer stage that stores the samples and second & third stages are correlation and accumulation stages respectively. Due to shared nature of this architecture, up to 92% less area is required as compared to direct implementation. Similarly, for Digital Audio broadcasting, [36] has implemented a system that can estimate and correct the frequency offset.

Timing and Frequency Synchronization scheme has been proposed and implemented in FPGA for IEEE standard 802.11a in [37]. Commonly used training samples have been deployed in order to measure the frequency offset in carrier using correlation. This correlation is calculated using the iterative architecture to save the area. Comparison has also been made between floating point simulation and fixed point implementation to validate the system.

Similarly, another implementation for wireless LAN modem with IEEE 802.11a standard has been done in [38]. C++ along with the help of hardware libraries have been used to define the architecture. Note that coarse frequency estimation is quite rough and does not calculate the exact frequency within tolerate able limits. For this purpose, fine frequency estimation algorithm is used.

2.7 Summary

This chapter forms the base for implementing the parts of SDR receiver. It contains basic relevant concepts related to digital signal processing, digital system design and computer architecture. It also discusses the coverage of system to be implemented along with its specifications, requirements and relevant calculations. Previous relevant research work both on algorithms and their relevant implementation on FPGAs is also the part of this chapter.

Chapter 3: Design Methodology

The implementation for various parts of SDR receiver has been discussed in this part of the document. All of the relevant designs, tables and illustrations along with arguments have been given in this chapter. Two different type of architectures are implemented for coarse frequency estimation. Most of the systems that are the part of SDR receiver are first simulated on MATLAB. On the basis of the written code, translation has been made to hardware architecture. Results obtained by these implementations are in discussed the next chapter.

3.1 Design & Implementation of CORDIC Algorithm

There are numerous amounts of factors that affect design aspects. One of the most important of them is the intrinsic property of this algorithm; output values are valid only for limited set of input values. The importance of this implementation lies in the fact that most of the communicational algorithms to be implemented are based on trigonometric and angular calculations of complex number.

To calculate the values out of the valid range, the support of trigonometric identities can be used. It can be implemented by first altering the input according to the trigonometric identity such that the value becomes within well calculable range for CORDIC Algorithm. After obtaining trigonometric result, the output can operate accordingly on that result to obtain the result corresponding to the actual input.

3.1.3. Major Top Level Parts of CORDIC System

In order to realize the solution of the problem discussed above, the system can be composed of the following parts.

- CORDIC Core
- CORDIC Input Interface
- CORDIC Output Interface

- Control Path

The part CORDIC core will implement the actual CORDIC algorithm. CORDIC Input and Output Interfaces will implement the trigonometric identities. Control path will have the sequence with which all operations will be performed. All of these parts have been discussed later in this chapter. Figure 3.1 shows the top level structure of CORDIC system.

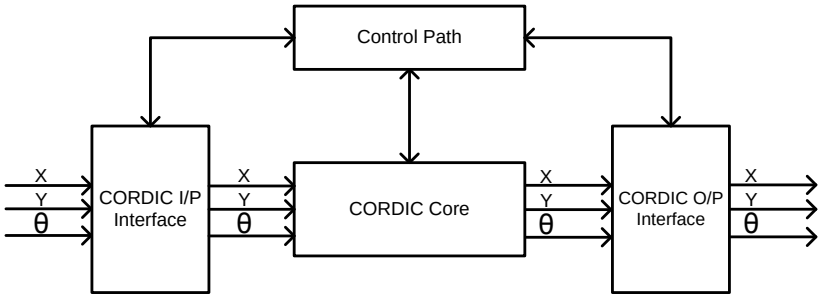

Figure 3.1 Top Level Flow Diagram of CORDIC Implementation

3.1.4. Selection of total Number of Iterations for CORDIC Algorithm

As mentioned by set of eq. 2.8 to eq. 2.10, CORDIC is an iterative algorithm. With the point of view of implementing this algorithm in FPGA, the total iterations as well as precision of processing variables is equally important. By simulating the CORDIC algorithm for most widely used number of iterations in MATLAB the effect of increasing iterations on error rate can be compared.

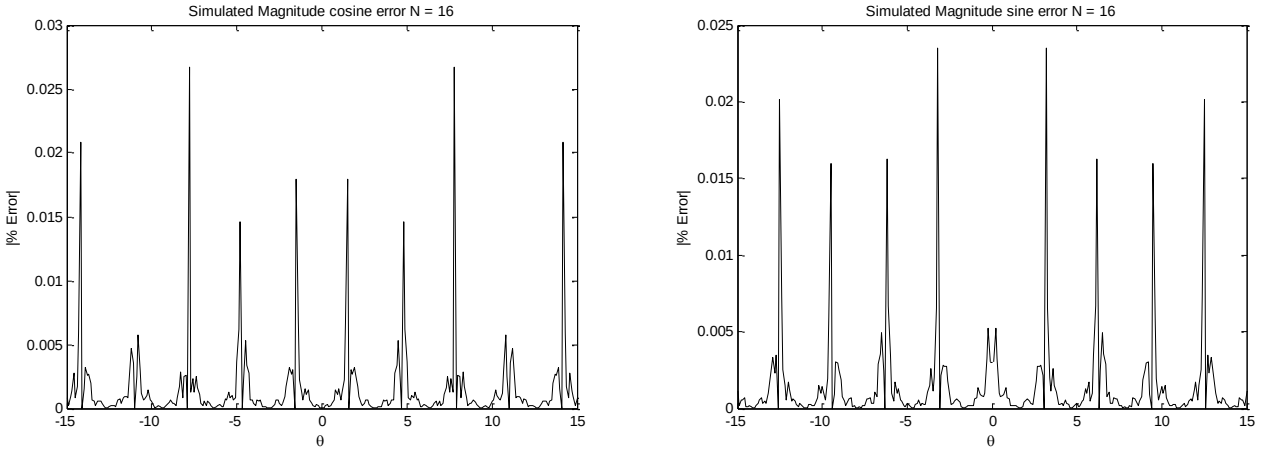

Figure 3.2 Calculated Magnitude Error in the simulation of CORDIC Algorithm (N = 16)

By comparing Figure 3.2 and Figure 3.3 it is quite evident that by increasing the total number of iterations for which this algorithm can operate decreases the percentage error. By surveying overall system and simulation results, the decision for total number of iteration is 32. The arguments supporting this design decision are following.

- The property of the targeted system is required to be optimized for accuracy; the reason of this is the fact that there is multiple numbers of algorithms that call the services of this system multiple times for the demodulation of single received burst. So the error gets accumulated each time received burst is passed through this system.
- The simulated results show that the percentage error for 32 iteration algorithm is at least 100,000 times less than that of 16 iteration algorithm.
- Although the area requirement may increase, this design decision also is made because total number of resources actually available in the targeted system is very high as evident from the Table 2.4.

As the current implementation can also be used for other algorithms and the throughput requirement may vary, parallelism can be used to satisfy this requirement.

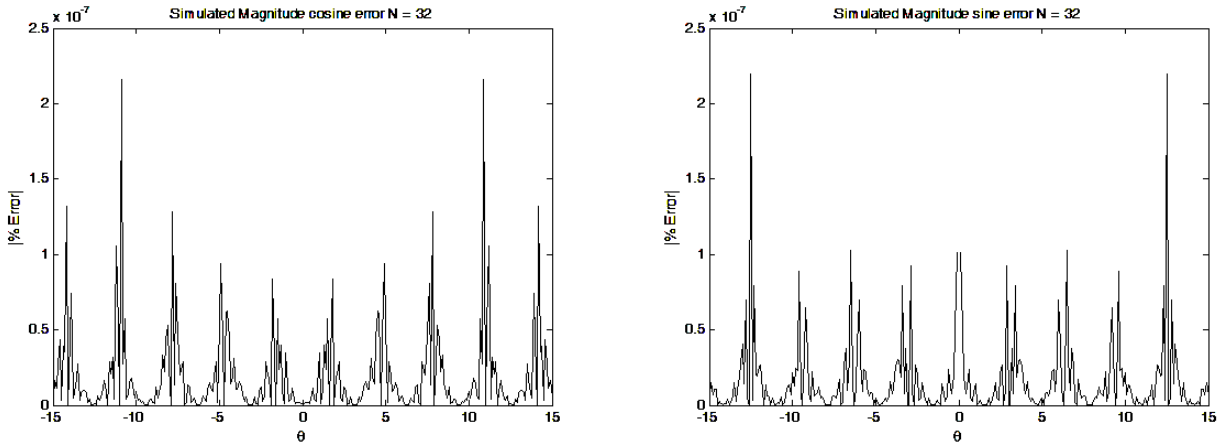

Figure 3.3 Calculated Magnitude Error in the simulation of CORDIC Algorithm (N = 32)

3.1.5. System Capabilities

Careful examination of coarse frequency estimation/compensation, channel estimation, fine frequency estimation/compensation and phase estimation it is evident that the system should be

able to calculate the trigonometric functions sine and cosine (Rotation mode). Also, system should be able to find the angle of complex number (Vectoring Mode).

3.1.6. Integration of Multiple Modes in Single System

Both the rotation and vectoring modes have remarkable similarities between them as shown in Table 2.5. The design decision for the implementation is to integrate both modes in single system. System will be able to change the mode in runtime. This decision has following supporting arguments.

- At any given time there will be only one mode (either rotation or vectoring) is required in runtime throughout the operation at the receiver end.
- It will decrease the required area and increase the area efficiency.

3.1.7. Usage of Trigonometric Identities to Solve Convergence Problem

In order to solve the convergence problem, trigonometric identities are employed. For rotation mode i.e. for the calculation of sine and cosine functions, the design decision is to use CORDIC core only for $-\frac{\pi}{4} \leq \theta \leq \frac{\pi}{4}$ to calculate the trigonometric functions. The selected range lies well within the correct calculable range of CORDIC core which lies from -0.9573 to 0.9573 radians to be exact. If θ is outside the selected range, trigonometric identities simplifies the problem as shown in Table 3.1.

Case	Output
$\frac{\pi}{4} < \theta \leq \frac{\pi}{2}$	$sin\theta = \cos\left(\frac{\pi}{2} - \theta\right)$ $cos\theta = \sin\left(\frac{\pi}{2} - \theta\right)$
$\frac{\pi}{2} < \theta \leq \frac{3\pi}{4}$	$sin\theta = \cos\left(\theta - \frac{\pi}{2}\right)$ $cos\theta = -\sin\left(\theta - \frac{\pi}{2}\right)$
$\frac{3\pi}{4} < \theta \leq \pi$	$sin\theta = \sin(\pi - \theta)$ $cos\theta = -\cos(\pi - \theta)$
$-\frac{\pi}{4} > \theta \geq -\frac{\pi}{2}$	$sin\theta = -\cos\left(-\frac{\pi}{2} - \theta\right)$ $cos\theta = -\sin\left(-\frac{\pi}{2} - \theta\right)$
$-\frac{\pi}{2} > \theta \geq -\frac{3\pi}{4}$	$sin\theta = -\cos\left(\theta + \frac{\pi}{2}\right)$

	$cos\theta = \sin\left(\theta + \frac{\pi}{2}\right)$
$-\frac{3\pi}{4} > \theta \geq -\pi$	$sin\theta = \sin(-\pi - \theta)$ $cos\theta = -\cos(-\pi - \theta)$
$-\frac{\pi}{4} \leq \theta \leq \frac{\pi}{4}$	$sin\theta = \sin(\theta)$ $cos\theta = \cos(\theta)$

Table 3.1 Calculation of sine and cosine using CORDIC algorithm

For the vectoring mode of CORDIC operation the requirement is to calculate the angle of complex number that lies in any quadrant. CORDIC algorithm gives correct result for $\tan^{-1}\left(\frac{y}{x}\right)$ if $-1.42 \leq \left(\frac{y}{x}\right) \leq 1.42$. It is required by the system that it should be able to calculate the correct angle to which complex number $x + yi$ points. In order to achieve this property, the design decision is to keep the ratio less than 1 by inverting it if it is greater. It is then provided to CORDIC core for processing. To get the accurate output, pre and post processing is done that is based on the actual input. The equivalent output for each of the case is shown in the Table 3.2.

Case	Subcase	Equivalent Output
x < 0, y <0	|x| > |y|	$\tan^{-1}\left(\frac{\|y\|}{\|x\|}\right) - \pi$
	|x| = |y|	$\frac{-3\pi}{4}$
	|x| < |y|	$-\tan^{-1}\left(\frac{\|x\|}{\|y\|}\right) - \frac{\pi}{2}$
x < 0, y =0	|x| > |y|	π
	|x| = |y|, |x| < |y|	Scenario Not Possible
x < 0, y >0	|x| > |y|	$-\tan^{-1}\left(\frac{\|y\|}{\|x\|}\right) + \pi$
	|x| = |y|	$\frac{3\pi}{4}$
	|x| < |y|	$\tan^{-1}\left(\frac{\|x\|}{\|y\|}\right) + \frac{\pi}{2}$
x = 0, y <0	|x| > |y|, |x| = |y|	Scenario Not Possible
	|x| < |y|	$\frac{-\pi}{2}$
x = 0, y =0	|x| > |y|	Scenario Not Possible
	|x| = |y|	0
	|x| < |y|	Scenario Not Possible
x = 0, y >0	|x| > |y|, |x| = |y|	
	|x| < |y|	$\frac{\pi}{2}$

x > 0, y <0	|x| > |y|	$-\tan^{-1}\left(\frac{\|y\|}{\|x\|}\right)$
	|x| = |y|	$\frac{-\pi}{4}$
	|x| < |y|	$\tan^{-1}\left(\frac{\|x\|}{\|y\|}\right)-\frac{\pi}{2}$
x > 0, y =0	|x| > |y|	0
	|x| = |y|, |x| < |y|	Scenario Not Possible
x > 0, y >0	|x| > |y|	$\tan^{-1}\left(\frac{\|y\|}{\|x\|}\right)$
	|x| = |y|	$\frac{\pi}{4}$
	|x| < |y|	$-\tan^{-1}\left(\frac{\|x\|}{\|y\|}\right)+\frac{\pi}{2}$

Table 3.2 Calculation of $\tan^{-1}(y/x)$ using CORDIC Algorithm (see text)

3.1.8. CORDIC Core Implementation

This section describes the actual implementation of CORDIC Algorithm shown in Figure 3.4. The implemented architecture is of iterative type. This will generalize the design so that any number of iterations is possible with least amount of changes. Only 3 registers each of 32 bits is required for this design and one iteration completes in one clock cycle which means it will take at least 32 clock cycles to complete the operation of this core.

Most important thing is the actual execution of single iteration. As given in the algorithm, initialization is necessary every time calculation starts. So, initialization for both the modes in the same system can be done by implementing four to one multiplexers as shown in the implemented design. The behavior of the system on the basis of the control pins of these muxes are shown in the Table 3.3.

Control Pins	System Behavior
2'b00	Iteration is in progress in Rotation Mode
2'b01	Initialization of x and y with appropriate constants in Rotation Mode. While z is equated to the θ for which trigonometric functions are to be calculated.
2'b10	Iteration is in progress in Vectoring mode
2'b11	Initialization of z in Vectoring mode. While y and x are initialized with values for which inverse tangent is required to be calculated.

Table 3.3 CORDIC Core Beahivour

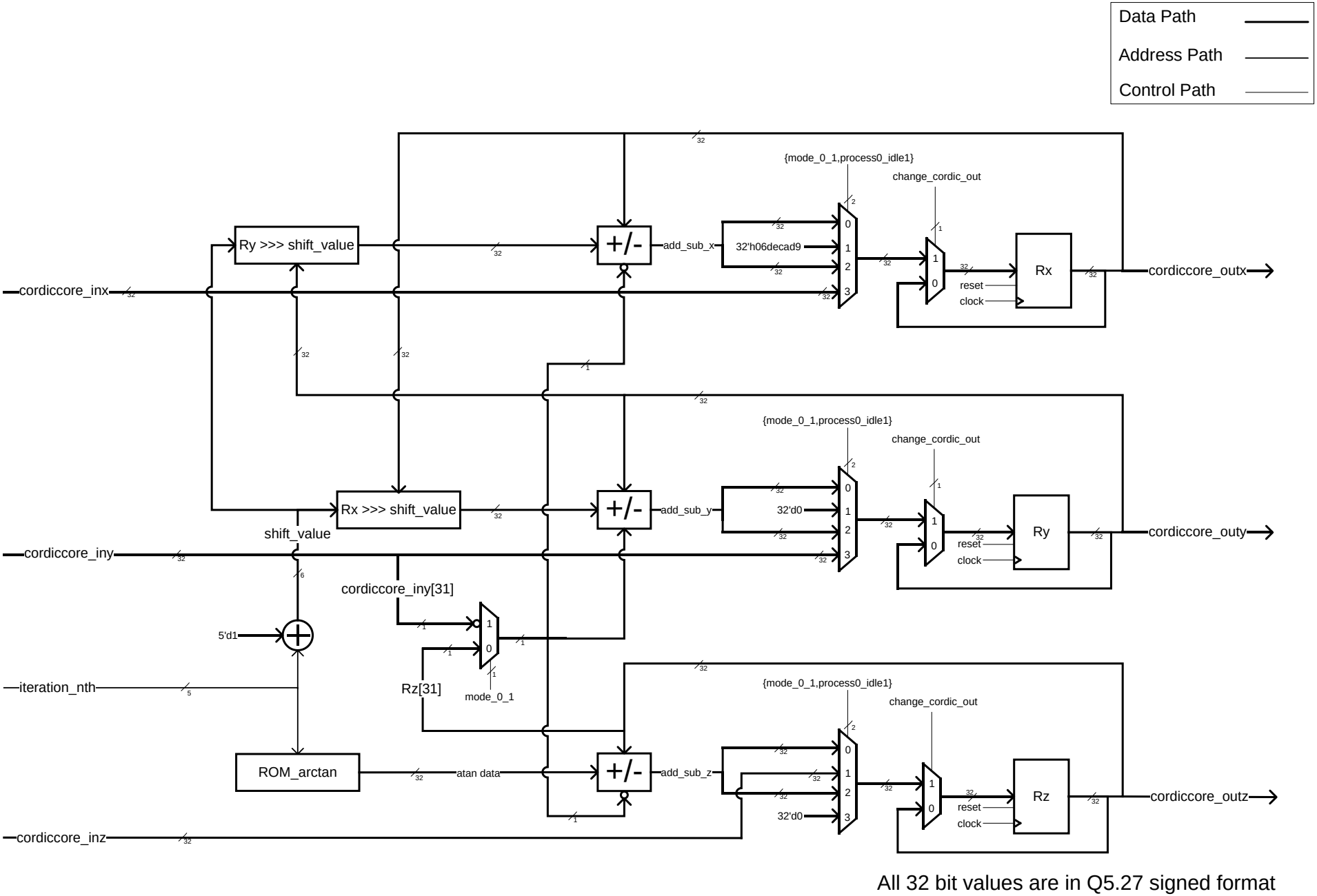

Figure 3.4 CORDIC Core Datapath Implemented Design

The two to one mux, whose output is the input of adders, serves the purpose of the testing condition which decides for addition/subtraction in two different modes. In Rotation mode, the sign of z while in Vectoring mode the sign of y decides either the operation for next iteration is to be addition or subtraction. Input named as Iteration_nth is used for indexing a specific value from ROM_arctan. Moreover, it also gives amount of rotation for output of Rx and Ry. Its value ranges from 0 to 31 and gets incremented after single clock cycle.

The entity ROM_arctan shown in Figure 3.4 acts as a look up table in which already calculated values of $tan^{-1}2^{-i}$ required for CORDIC processing are stored (where i is from 1 to 32). So, for 32 iteration system, total number of locations in this look up table is also 32. Each location contains a 32 bit number in Q5.27 signed format. This is the same Q point format as used for the entire CORDIC core.

3.1.9. Input and Output Interfaces Implementation

Both of the input and output interfaces are designed together as both of them implements trigonometric identities as given in Table 3.1 and Table 3.2. The interfaces along with the CORDIC core are shown in Figure 3.5. On the arrival of input, the input interface decides the mode of operation and also informs to the output interface so that correct value for the actual input can be calculated. The input interface has the two parts; one is dedicated for rotation while other is for vectoring mode. For each part, the mode of operation is selected on the basis of the quadrant in which input lies as illustrated in Figure 3.5.

The input interface basically consists of series of comparators. The only difference in these parts is at any mode only one of them is in the operation. The output interface of the system is common for both of the modes as some common nature of operations are required at this point. The output interface has three 32 bit registers that stores the final calculated value. Values in eight by one multiplexer are $\frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi, \frac{-\pi}{4}, \frac{-\pi}{2}, \frac{-3\pi}{4}, -\pi$ written in Q5.27 signed format.

Likewise CORDIC core, the Q point format of both input and output interface is Q5.27 signed. One reason of this Q point selection is that, it gives enough high precise output and the range for which the calculation can be done in both modes is moderately high; the system shown in Figure 3.5 has the capability to calculate sine and cosine from -5pi to 5pi in rotation mode.

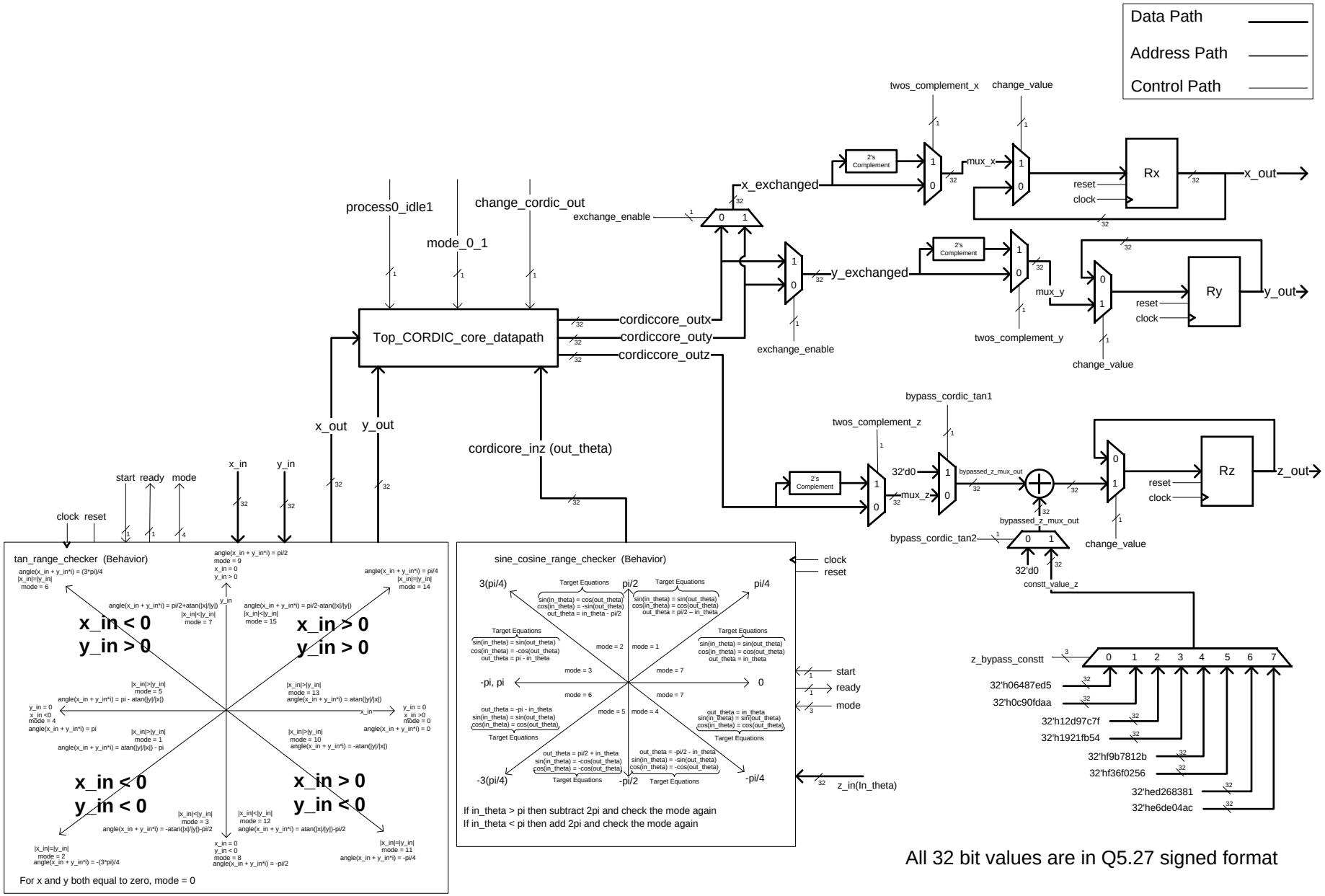

Figure 3.5 Main CORDIC Datapath Design

3.1.10. Control Path Implementation

All of the sequences of operations to be performed as discussed in previous sections are controlled by this part. It is simply a state machine with multiple states each generating specific control signals. The implemented behavior is shown in Figure 3.7. To start processing one of the rotation or vectoring mode is selected, values for which processing is required to be placed at the input and start pin is triggered. After the processing is complete and the output is ready, this state machine asserts the out_ready pin.

3.2 Coarse Frequency Estimation Implementation

Implementation of coarse frequency estimation algorithm is given in eq. 2.14. Its implementation is shown in Figure 3.10 and Figure 3.12. This equation can be divided into two major parts. A factor $b\left(-\frac{N}{2}+k+1\right).\left(\frac{a(k+1)}{a(k)^*}\right)^*$ is one dimensional array that will remain constant over the entire period of processing. Thus this portion of equation can be calculated in offline mode. A second major part consists of real time data input required for processing i.e. $r(k+1).r(k)$.

The approach of implementation is to calculate both of these parts separately, multiply the result then calculate the angle using CORDIC implementation. Both factors of first part are already known. The factor b(k) can be calculated from the eq. 2.11 while a(k) is the noiseless transmitted training sequence which is shown in Figure 3.6.

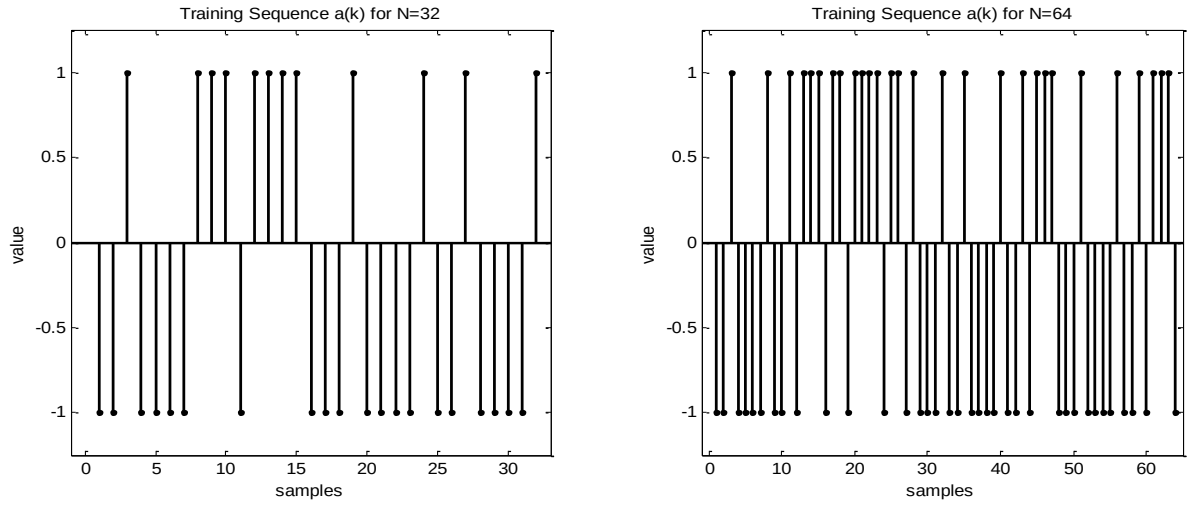

Figure 3.6 Training Sequence to be Transmitted

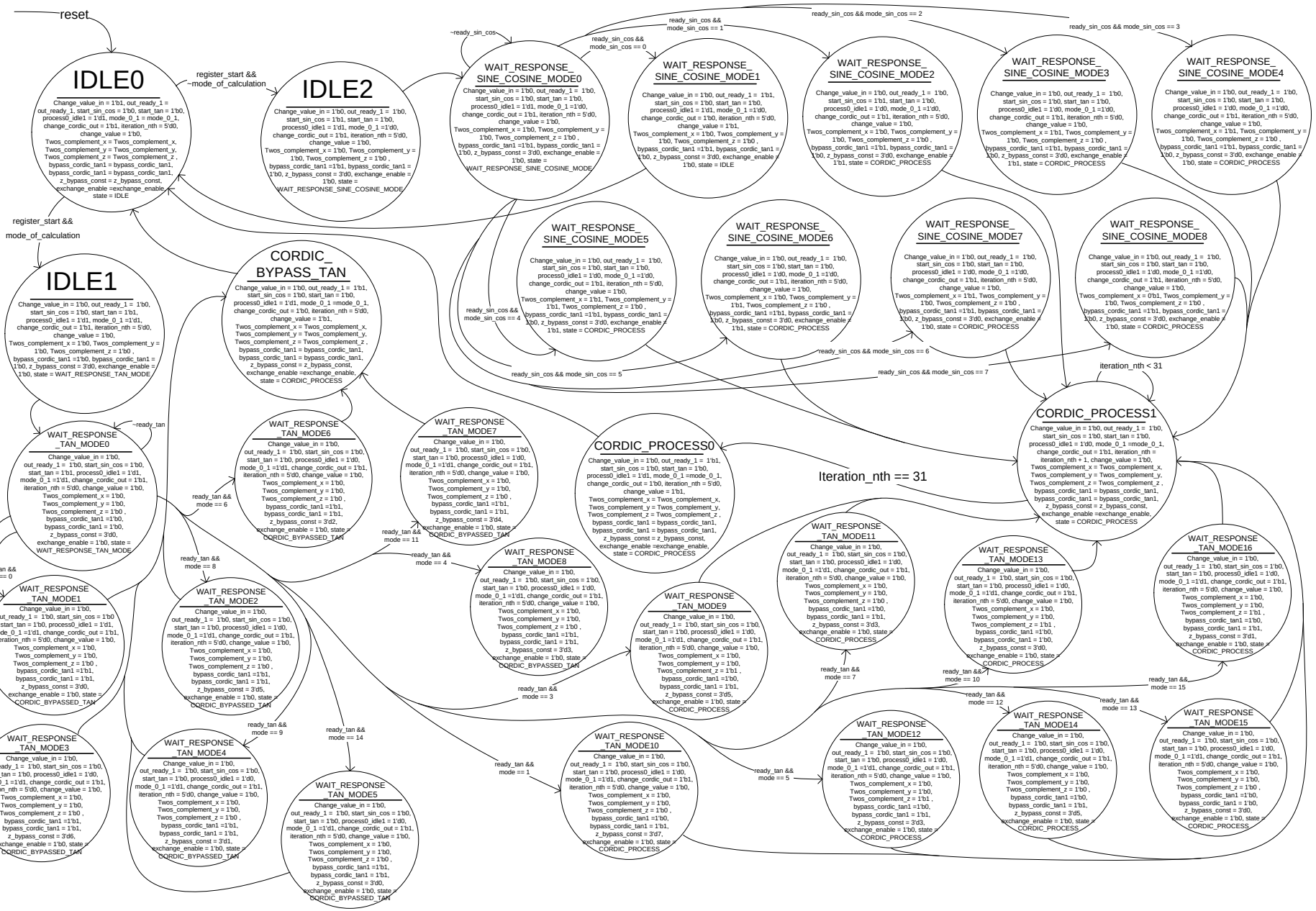

Figure 3.7 Top Level CORDIC Control Path (with control signals)

As shown in Figure 3.6, there are two different lengths of training sequences depending upon the system configuration. After calculating the result on MATLAB the result is illustrated in Figure 3.8a

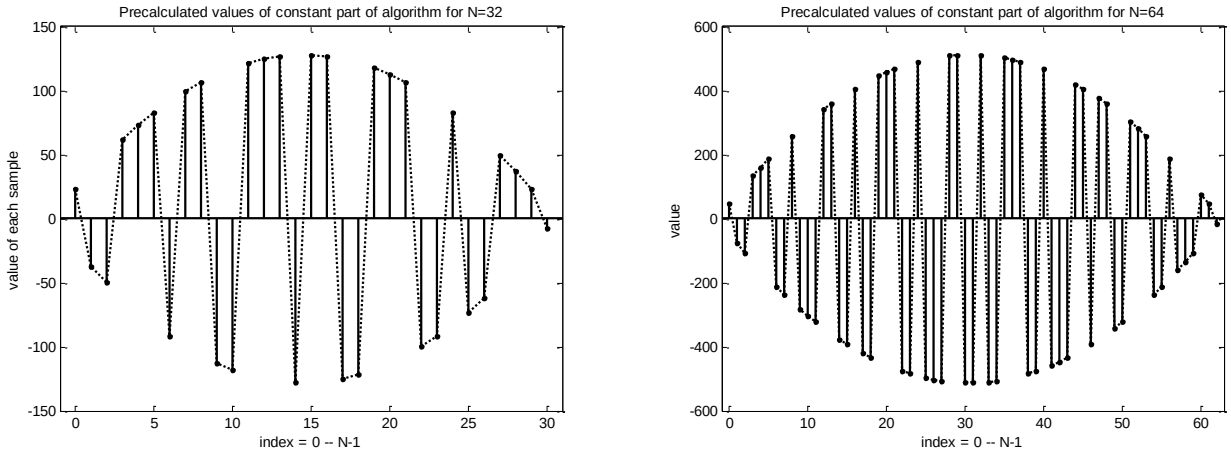

Figure 3.8 Coarse Frequency Estimation Algorithm Pre Calculable Result

In order to save this result, 1.5 kilobits of memory is required (if every sample is represented in 16 bits). But this amount of resource can be drastically reduced if the pattern hidden in the sequence is figured out. By carefully observing the results, it can be concluded that the difference between absolute values of consecutive samples is linear and easily generate able. So, if only first value of the sequence is known, the rest of values can be generated. For illustration, the difference of consecutive samples has been shown in Figure 3.9.

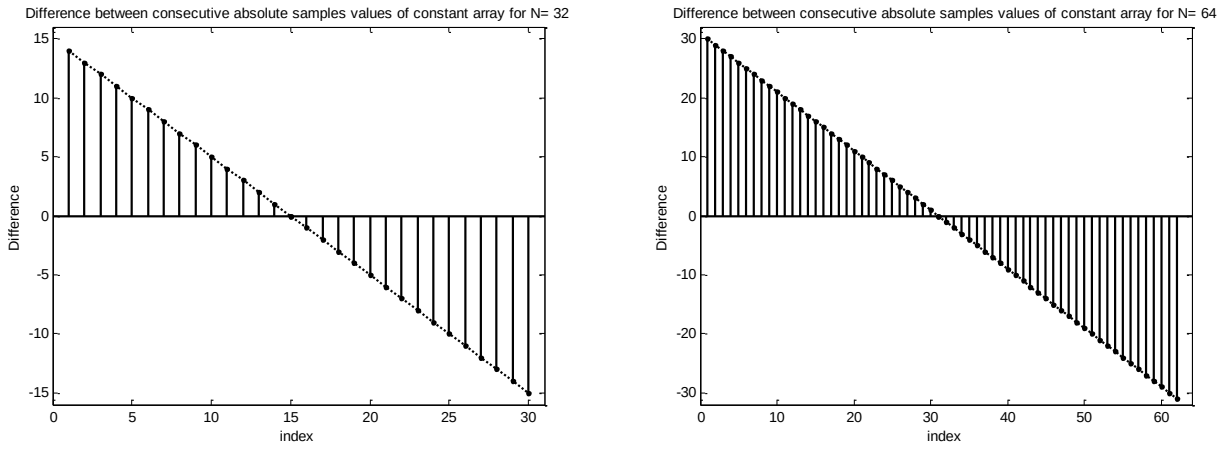

Figure 3.9 Absolute Difference between the Consecutive Samples of the result shown in Figure 3.8

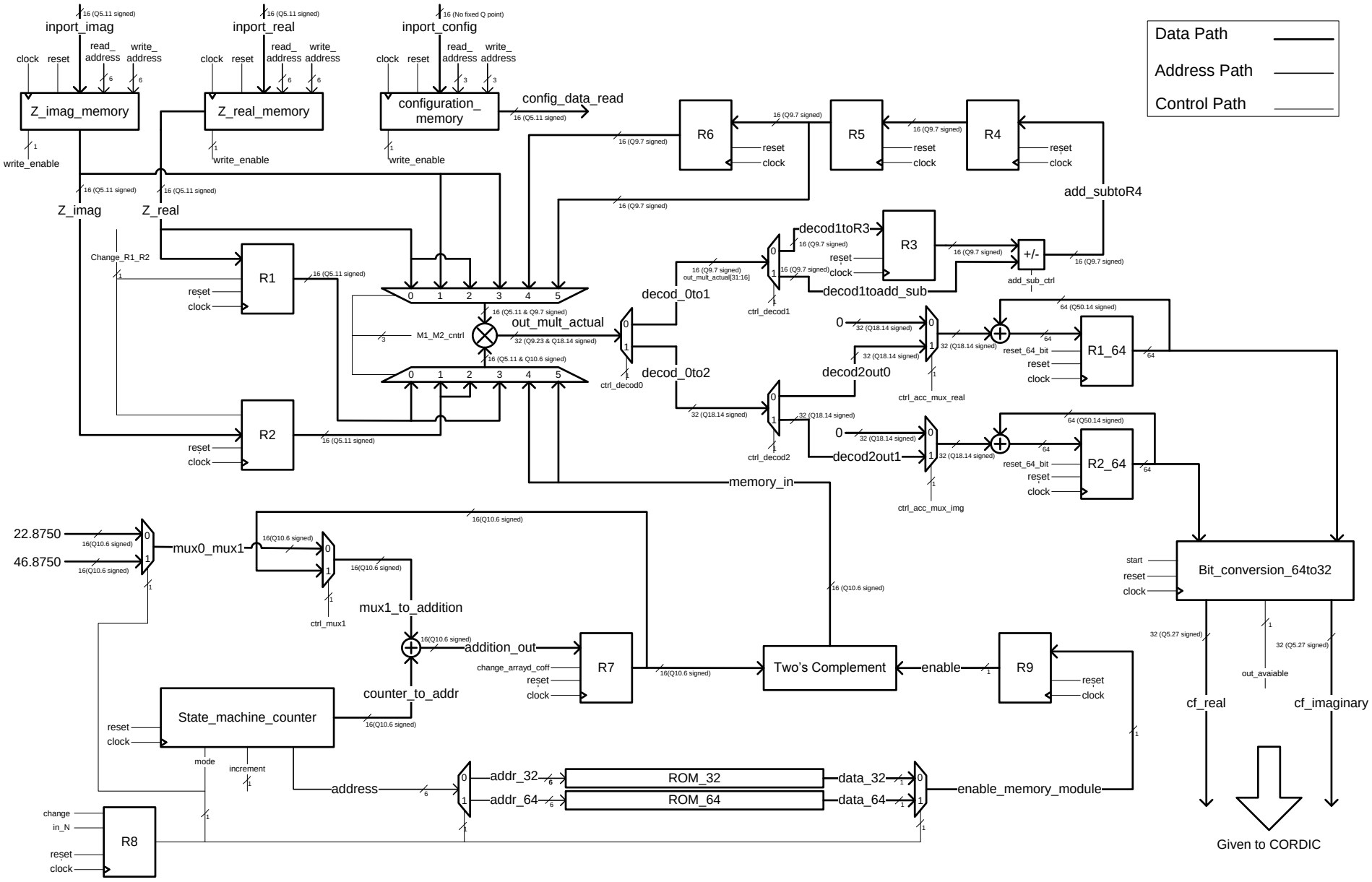

Figure 3.10 Shared Resource Coarse Frequency Estimation Datapath

3.2.3. Shared Resource Coarse Frequency Estimation Datapath

The design of datapath is optimized for efficient resource utilization. Only one multiplier is enough to handle the data for data rate given in Figure 2.15. For each value of the control signal M1_M2_cntrl, the signals used as a multiplicand in the multiplication performed by six to one multiplexers is shown in the Table 3.4.

Value of M1_M2_cntrl	Signals as Multiplicand
0	$\mathrm{Real}(\mathrm{r}(\mathrm{k})) * \mathrm{Real}(\mathrm{r}(\mathrm{k}+1))$
1	$\mathrm{Imag}(\mathrm{r}(\mathrm{k})) * \mathrm{Imag}(\mathrm{r}(\mathrm{k}+1))$
2	$\mathrm{Imag}(\mathrm{r}(\mathrm{k})) * \mathrm{Real}(\mathrm{r}(\mathrm{k}+1))$
3	$\mathrm{Real}(\mathrm{r}(\mathrm{k})) * \mathrm{Imag}(\mathrm{r}(\mathrm{k}+1))$
4	$\mathrm{Real}\big(\mathrm{r}(\mathrm{k}) * \mathrm{r}(\mathrm{k}+1)\big) * Precalculable\_Part(k)$
5	$\mathrm{Imag}\big(\mathrm{r}(\mathrm{k}) * \mathrm{r}(\mathrm{k}+1)\big) * Precalculable\_Part(k)$

Table 3.4 Complex Multiplication Using Shared Muliplier

R3 to R6 registers hold values calculated when M1_M2_cntrl is 0 to 3. Once the required result is calculated (when M1_M2_cntrl is 4 and 5), the real and imaginary part of the result is accumulated in R1_64 and R2_64 respectively. Both of these registers are of 64 bit while all others in current design are of 16 bit precision.

The pre calculable part is saved in the design portion where state machine counter is located. This counter is used to generate the consecutive values shown in Figure 3.9. ROM_32 and ROM_64 tell the sign of each computation. When the value of N is 32, the counter generates from 14 to -15; decrementing one in each clock cycle. The generated value is then added to the initial value 22.8750 to get the absolute result. The sign of this result is stored in ROM_32 which is of 32 bits. It outputs one bit 0 or 1 for positive or negative sign respectively. Same is the case for N=64 in which counter generates from 30 to -31, initial value is 46.8750 and sign of each computation is stored in ROM_64. By using this strategy, the memory requirement is only 96 bits. The values of ROM_32 and ROM_64 are given in eq. 3.1 and eq. 3.2 respectively.

$$\boldsymbol{ROM\_32 = 32'h46C6\_4646} \quad \text{(3.1)}$$

$$\boldsymbol{ROM\_64 = 64'h4EC6\_4EC6\_CEC6\_CEC6} \quad \text{(3.2)}$$

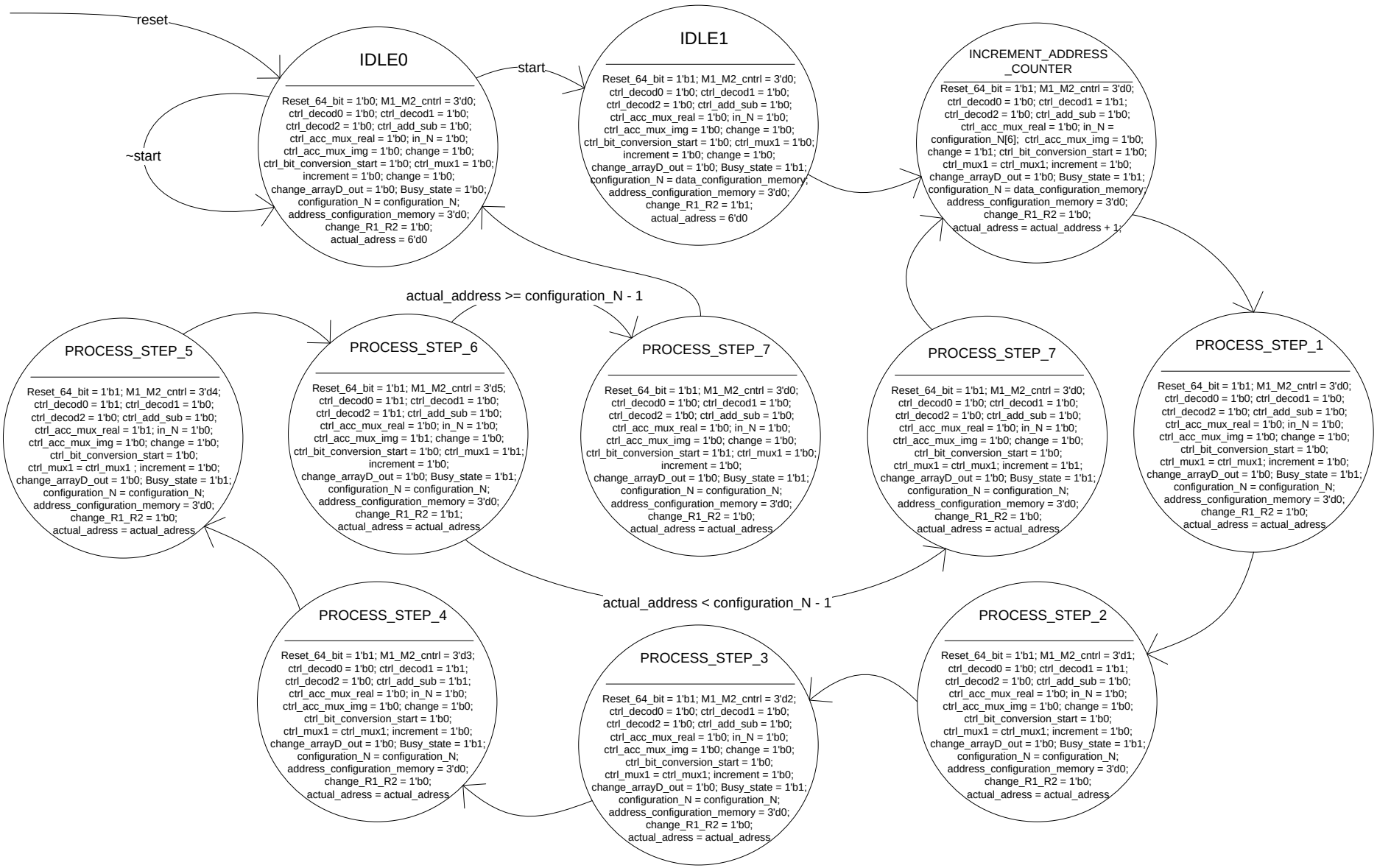

Figure 3.11 Coarse Frequency Estimation Control Path for optimized area implementation

3.2.4. Performance Optimized Coarse Frequency Estimation Datapath

Coarse frequency estimation data path optimized for high throughput is shown in Figure 3.12. Memory architecture is similar for both previously discussed and current design. High throughput is achieved by exploiting the pipelining and implementing complex multiplication using dedicated multipliers. As shown in figure, multiple pipeline cuts provide the opportunity to get maximum throughput from the system. Throughput calculation and its comparison with previous implementation is done in next chapter.

3.2.5. Coarse Frequency Estimation Control Path

The implemented state machine for shared type of implementation is shown in Figure 3.11. Once the start pin is triggered, it starts its iterative operation for given number of data inputs. Once those iterations are complete, it goes idle again while output ready pin is raised. State machine for Figure 3.12 is simpler than Figure 3.11 shown because no control signals for iteration are involved.

3.3 Summary

This chapter presents the design blueprints of SDR receiver implementations in FPGA. Discussion on these designs is done along with tables and figures for illustration purposes. Arguments on some of the important design decisions are given. Moreover, due to some pre known factors, some algorithms are simplified such that they can be implemented with fewer resources.

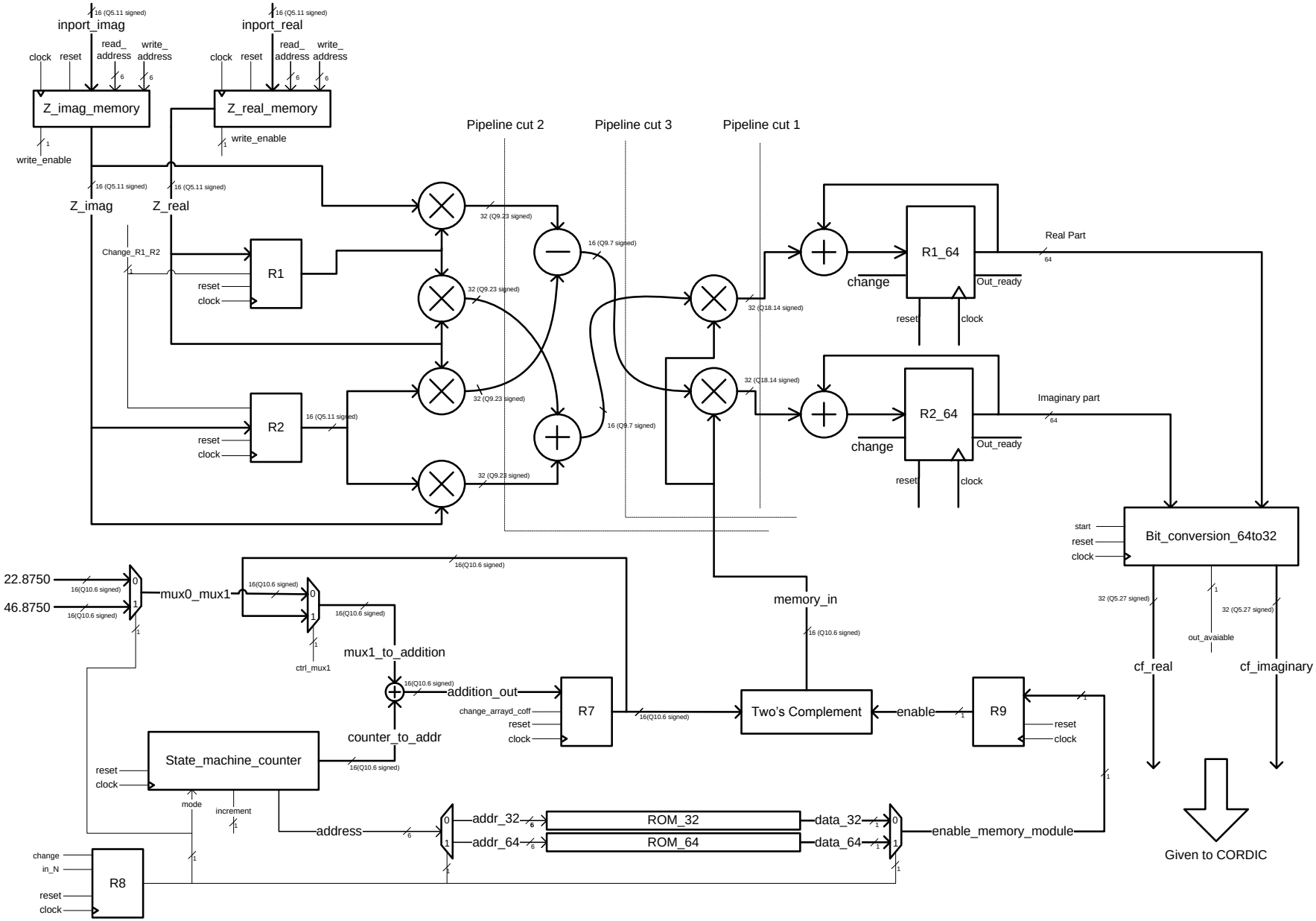

Figure 3.12 Performance optimized Coarse Frequency Estimation Datapath

Chapter 4: Results and Discussion

In this chapter, results of the implementations have been presented. It also contains the brief discussion on these results. These results include the percentage error, resource requirements, maximum achievable clock and print screens of simulations. The target device is Xilinx 3A DSP xc3sd3400a FPGA.

4.1 CORDIC Implementation Results

This section contains the detailed results obtained by the implementation of proposed CORDIC designs.

4.1.3. Percentage Error

Outcomes of percentage error for both (rotation and vectoring) modes of CORDIC operation are given in this section. Figure 4.1 and Figure 4.2 shows the percentage cosine and sine error respectively. While Figure 4.3 and Figure 4.4 shows the visual comparison of the values calculated in MATLAB and results of implementation.

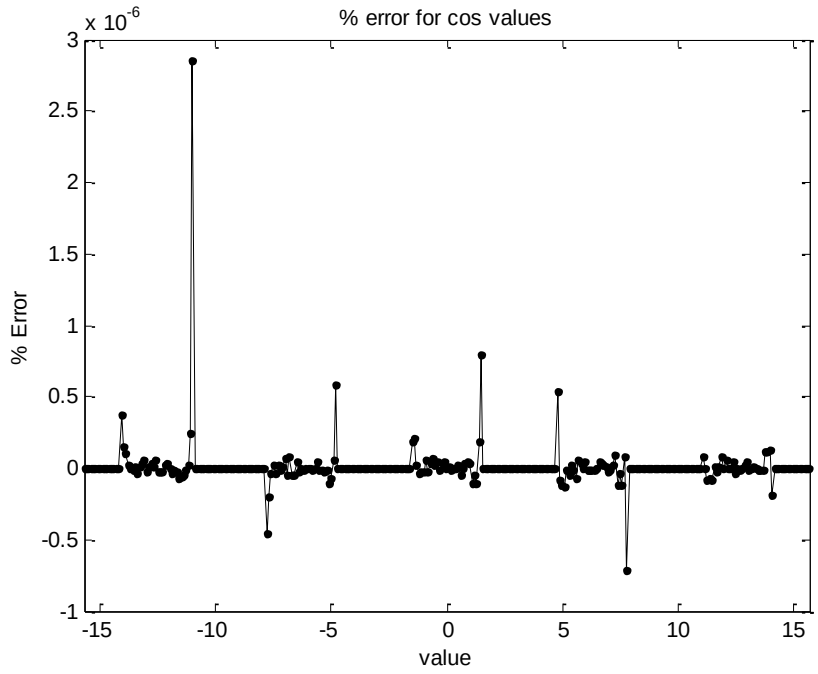

Figure 4.1 Percentage Cosine Error of 32 bit CORDIC Implementation (Rotation Mode)

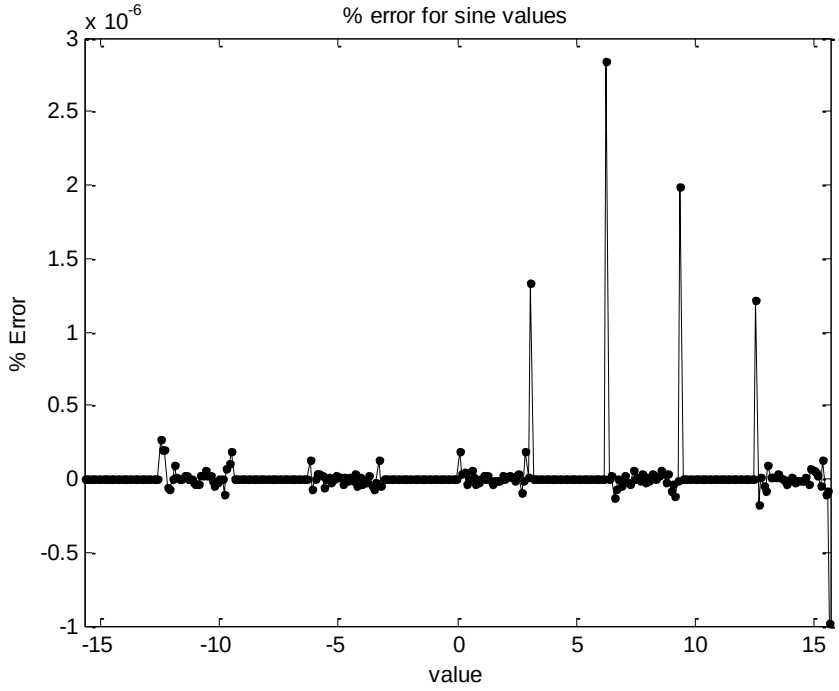

Figure 4.2 Percentage Sine Error of 32 bit CORDIC Implementation (Rotation Mode)

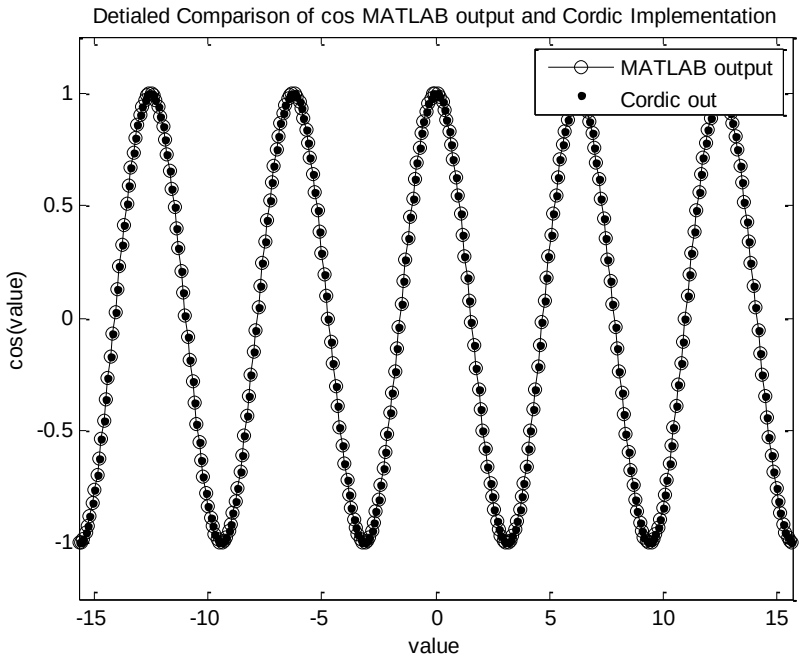

Figure 4.3 Visual Comparison of Results of CORDIC Calculation of Cosine with Corresponding MATLAB Output

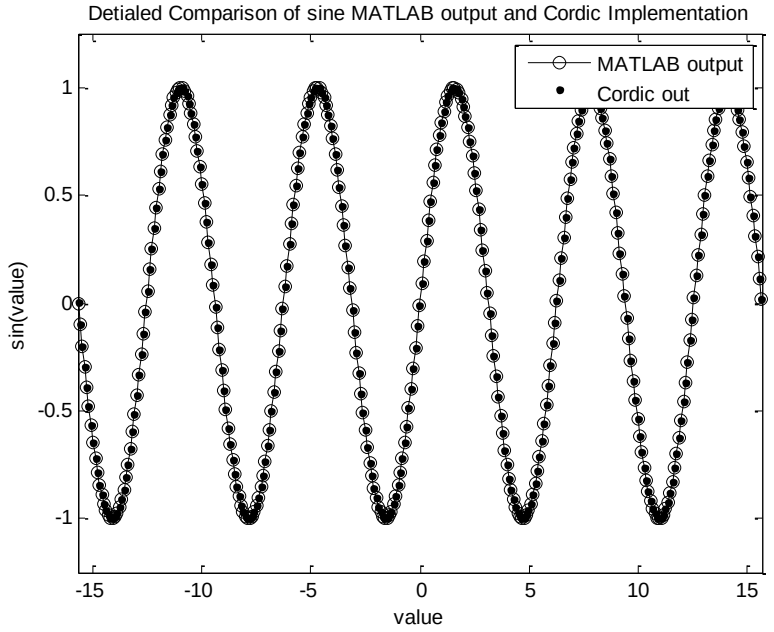

Figure 4.4 Visual Comparison of Results of CORDIC Calculation of Sine with Corresponding MATLAB Output

Another way of looking at these results is to draw a waveform by calculating values for rotation mode of CORDIC implementation and to compare the values with calculation done in MATLAB as shown in Figure 4.3 and Figure 4.4. All of these results have been calculated with input and output interfaces included in the design. Input values range from -15 to 15 after constant intervals.

For vector mode of operation, the values are selected such that system gets verified for large number of values lying in four quadrants that is both x and y range from -2 to 2. Results are shown in Figure 4.5 and visual comparison of values obtained in vector mode with MATLAB is shown in Figure 4.6. Mesh and surface represents the values calculated by MATLAB and vector mode of CORDIC implementation respectively.

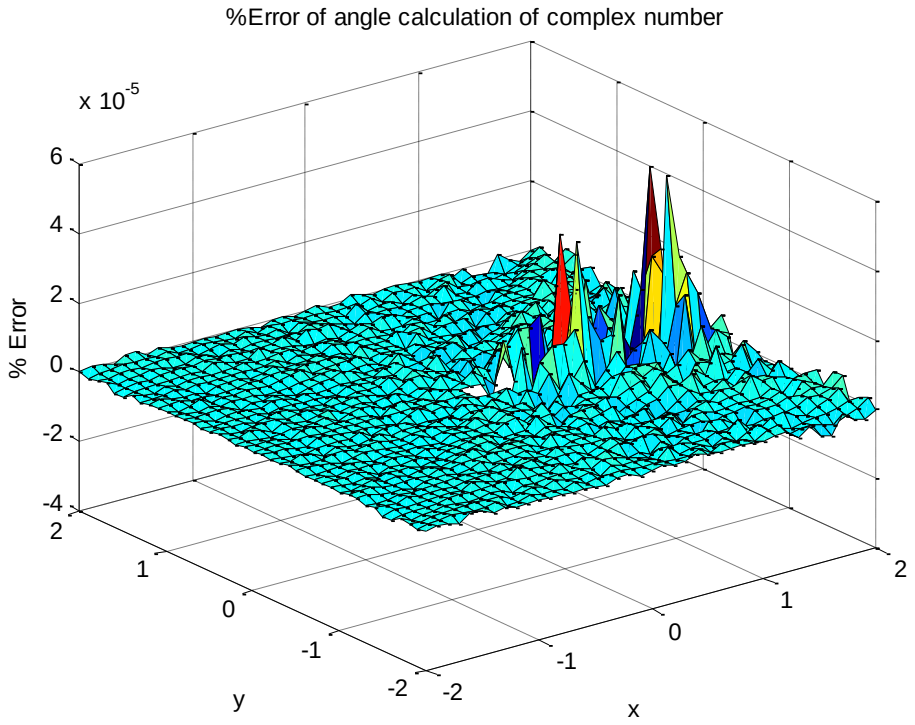

Figure 4.5 Percentage Error for Vectoring mode of CORDIC implementation

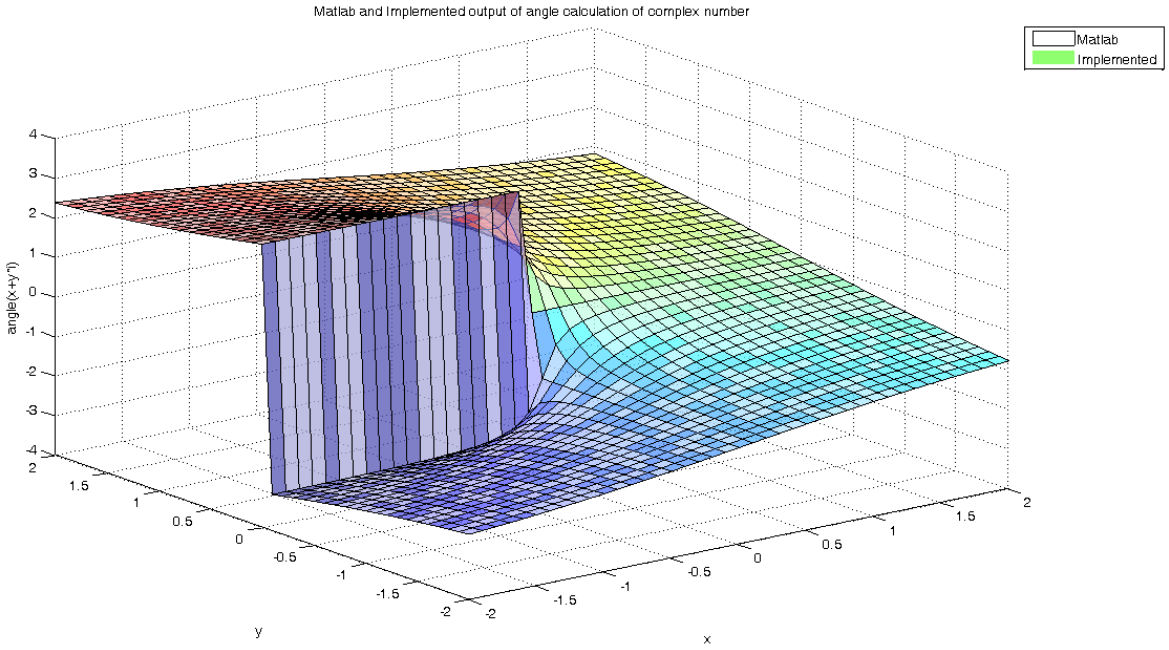

Figure 4.6 Visual Comparison b/w Values generated by Vectoring mode and MATLAB

4.1.4. Simulated Waveforms

Waveforms of the system for vectoring and rotation modes are shown in Figure 4.7and Figure 4.8. A simulated operation of Rotation mode of CORDIC is shown in above figure. It calculates the sine and cosine values for 10.808, -1.9960 and -1.8578 (all in Q5.27 signed format). All of these values are placed at z_in. Valid results are at {x_out, y_out} once out_ready is high and for these inputs, they are {-0.1865, -0.9825}, {-0.4125, -0.9110} and {-0.2831, -0.9591}. The format of output is {cos(input), sin(input)}. Average clock cycles required for calculation are 38.

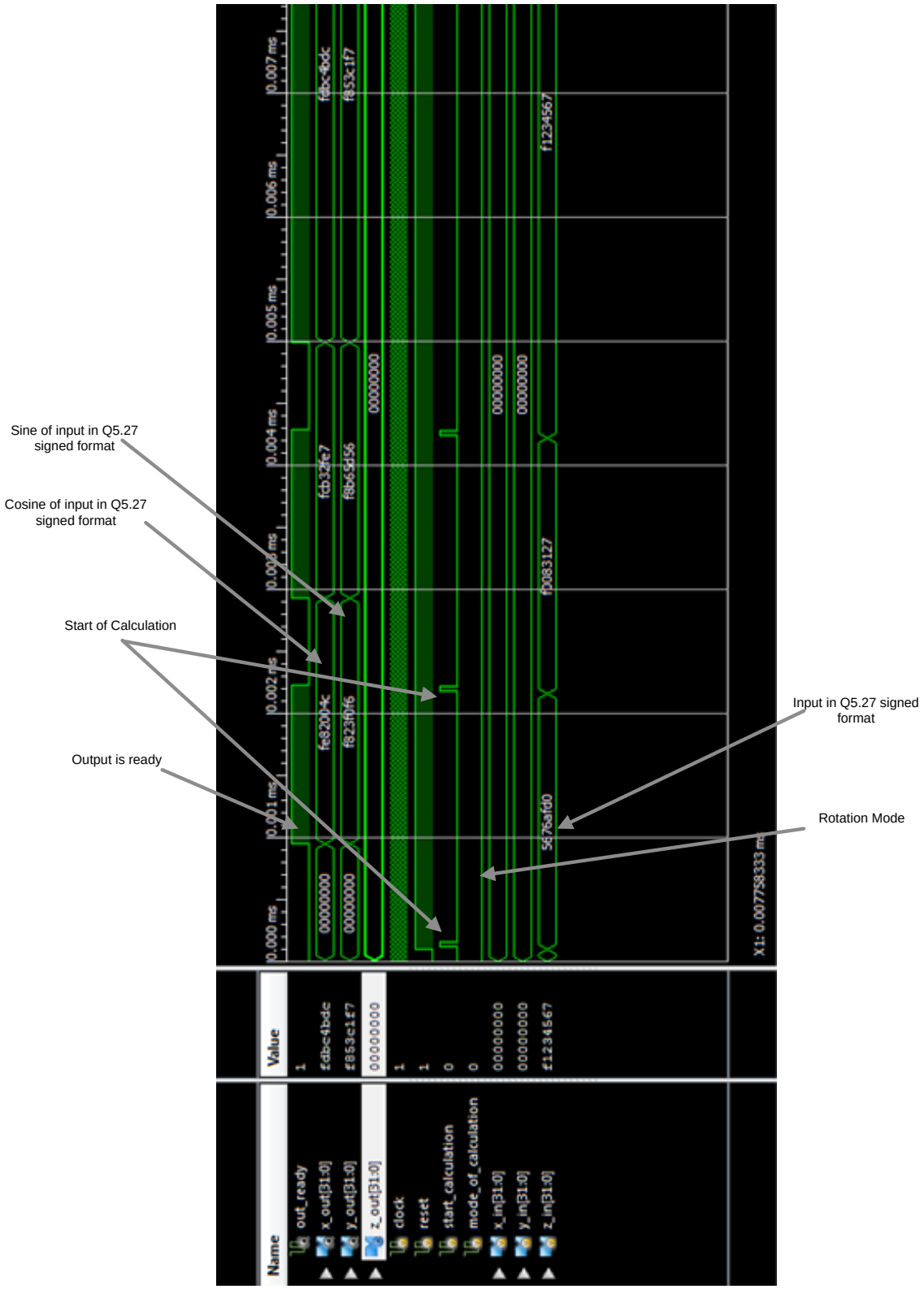

Figure 4.7 Simulated Waveform for Rotation mode of CORDIC

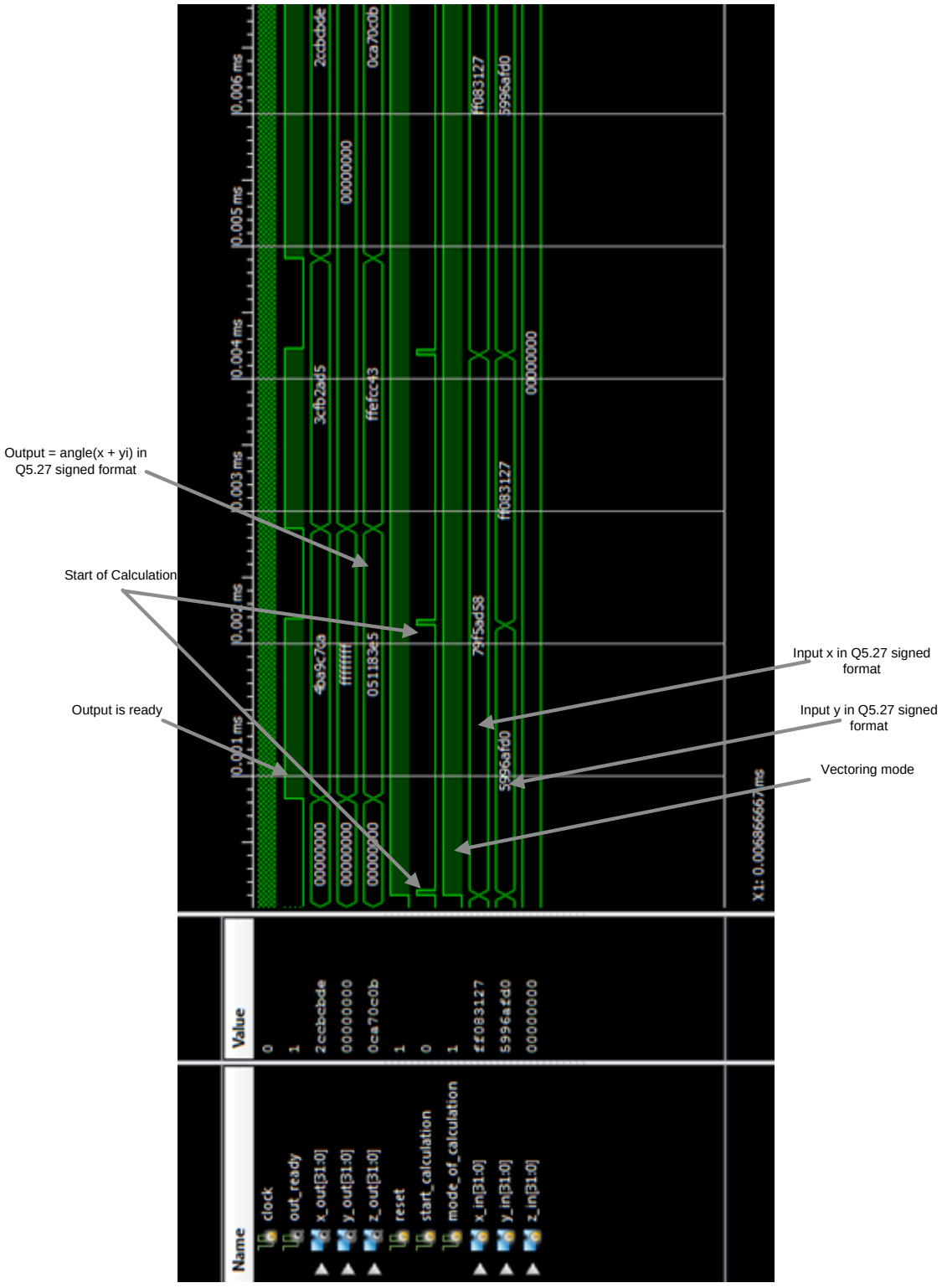

Figure 4.8 Simulated Waveform for Vectoring mode of CORDIC

In Figure 4.8, angle of three complex numbers have been calculated. They are 15.2450 + 11.1986i, 15.2450 – 0.1210i and -0.1210 + 11.1986i. All of these numbers are represented in Q5.27 signed format. Calculation takes on average 36 clock cycles and results are visible at z_out which are 0.6336, -0.0079 and 1.5816. Note that mode of calculation is 1 which represents vectoring mode.

4.1.5. Area Consumption and Timing Results

Resources consumed by various implemented parts are given in Table 4.1.

Component	Resources Consumed	Maximum Clock (Post Place and Route)
Core Datapath (Figure 3.4)	**Slice Flip Flops:** 96 (0.2%) **Total 4 Input LUTs**: 682 (1.42%) **Occupied Slices:** 393 (1.65%) **BUFGMUXs (Global Clock):** 1 (4.16%)	86.125 MHz
Input Interface for Rotation Mode (Figure 3.5 & Table 3.1)	**Slice Flip Flops:** 41 (0.086%) **Total 4 Input LUTs**: 539 (1.13%) **Occupied Slices:** 278 (1.16%) **BUFGMUXs (Global Clock):** 1 (4.16%)	61.778 MHz
Input Interface for Rotation Mode (Figure 3.5 & Table 3.2)	**Slice Flip Flops:** 39 (0.082%) **Total 4 Input LUTs**: 617 (1.29%) **Occupied Slices:** 311 (1.30%) **BUFGMUXs (Global Clock):** 1 (4.16%) **DSP48As:** 2 (1.59%)	131.874 MHz

Top Level Datapath (Figure 3.5)	**Slice Flip Flops:** 368 (0.77%) **Total 4 Input LUTs**: 2145 (4.50%) **Occupied Slices:** 1180 (4.94%) **BUFGMUXs (Global Clock):** 1 (4.16%) **DSP48As:** 2 (1.59%)	61.630 MHz
Top Level Control Path (Figure 3.7)	**Slice Flip Flops:** 27 (0.06 %) **Total 4 Input LUTs**: 80 (0.17 %) **Occupied Slices:** 44 (0.18 %) **BUFGMUXs (Global Clock):** 1 (4.16%)	215.750 MHz
Top Level CORDIC	**Slice Flip Flops:** 395 (0.827 %) **Total 4 Input LUTs**: 2183 (4.57 %) **Occupied Slices:** 1198 (5.02 %) **BUFGMUXs (Global Clock):** 1 (4.16%) **DSP48As:** 2 (1.59%)	53.746 MHz

Table 4.1 Resource Consumed by CORDIC at various levels

4.1.6. Comparison with Previous Implementations

The percentage error of current implementation is much lower than its previous counterparts which are discussed in 2.6.3.1 . This is due to the fact that current implementation is of 32 bit and most of the previous implementations are 16 bit. Although the area consumption of current implementation is higher, it has better overall utilization of resources; multiple modes are implemented in single design with ability of convergence for larger range of input. Generally,

implementations of CORDIC are requirement specific and it's a tradeoff between different design parameters.

4.1.7. Maximum Throughput Calculation

Maximum achievable throughput for CORDIC implementation is calculated in eq. 4.1.

$$\text{Clock of System} = 53.746 \text{ MHz}$$

(Verified with clock constraint 17 ns 50 % duty cycle)

$$\text{Average Number of Clock Cycles required per output} \approx 38$$

$$\text{Maximum Throughput} = \frac{53.746\text{M}}{38} = 1.41 \text{ Msps}$$

$$= \mathbf{1.41\ Msps} \times \mathbf{32\ bits\ per\ sample} = \mathbf{45.12\ Mbps} \text{ - - - - - - - - - - - - (4.1)}$$

4.2 Coarse Frequency Estimation Implementation Results

In this section, all the relevant results related to Implementation of Coarse Frequency Estimation Algorithm.

4.2.3. Percentage Error for shared resource design

Percentage error calculated in implementation for training sequence of length 32 and 64 are shown in Figure 4.9 and Figure 4.10.

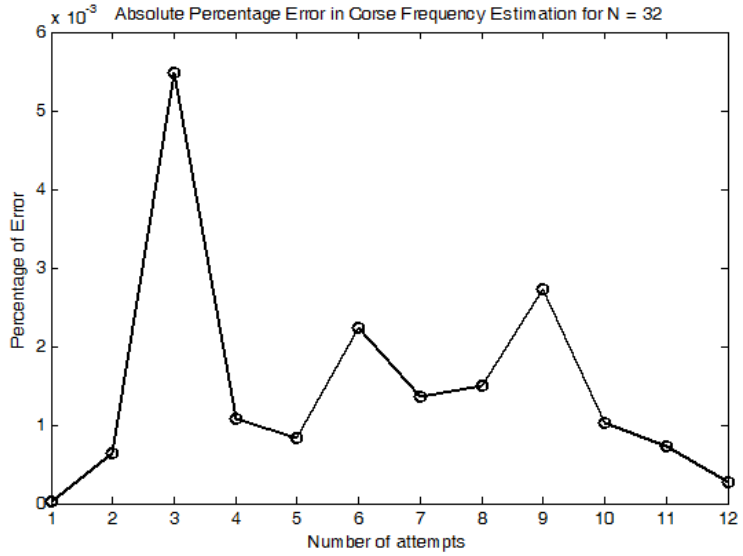

Figure 4.9 Percentage Error of Shared Resource Coarse Frequency Estimation for N=32

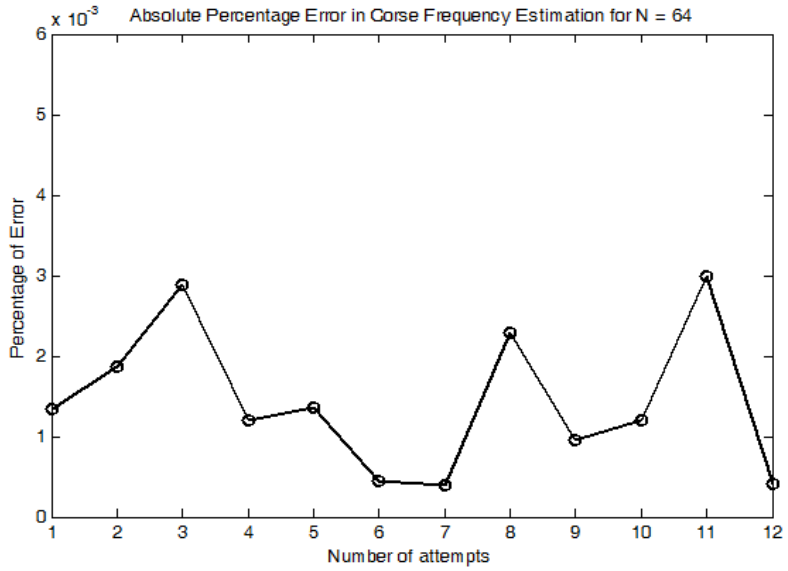

Figure 4.10 Percentage Error of Shared Resource Coarse Frequency Estimation for N=64

4.2.4. Percentage Error for Performance optimized design

Percentage error for performance optimized implementation is almost similar for shared resource. Figure 4.11and Figure 4.12 gives the percentage error of training sequence of length 32 and 64 respectively.

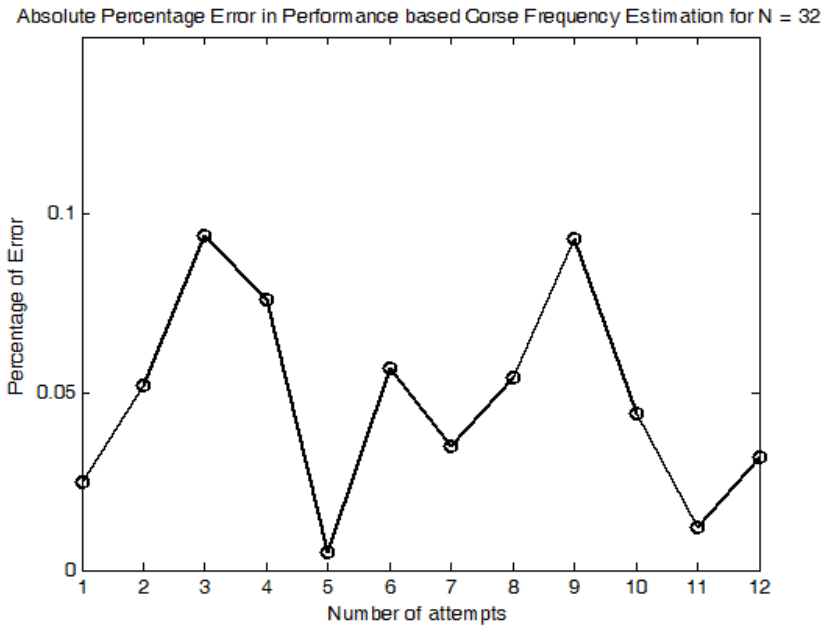

Figure 4.11 Percentage Error of Performance Based Coarse Frequency Estimation for N=32

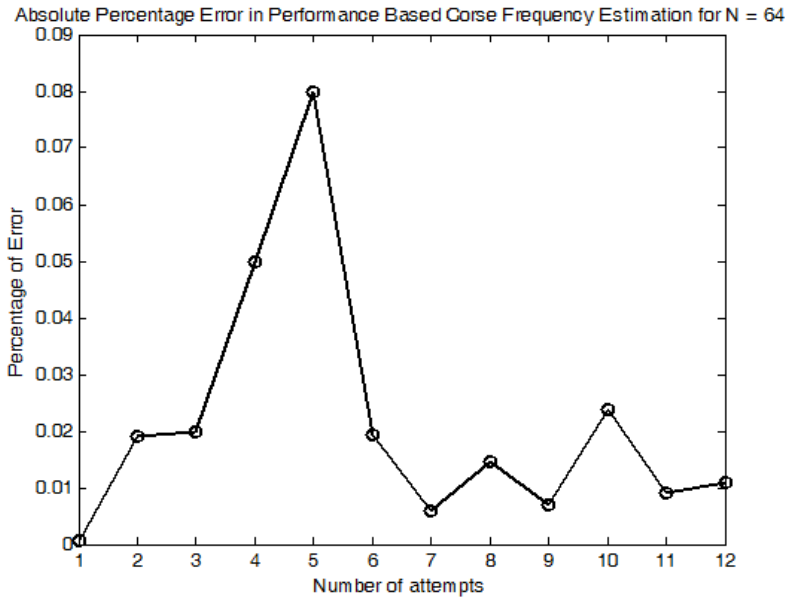

Figure 4.12 Percentage Error of Shared Resource Coarse Frequency Estimation for N=64

4.2.5. Area Consumption and Timing Results for shared resource design

Details of area consumed by both designs of Coarse Frequency Estimation system and maximum achievable clock are given in Table 4.2.

Component	Resources Consumed	Maximum Clock (Post Place and Route)
Data Path (Figure 3.10)	**Slice Flip Flops:** 394 (0.825 %) **Total 4 Input LUTs**: 659 (1.38 %) **Occupied Slices:** 433 (1.81 %) **BUFGMUXs (Global Clock):** 1 (4.16%) **DSP48As:** 1 (0.794 %)	56.815 MHz
Control Path (Figure 3.11)	**Slice Flip Flops:** 47 (0.1 %) **Total 4 Input LUTs**: 76 (0.16 %) **Occupied Slices:** 50 (0.21 %) **BUFGMUXs (Global Clock):** 1 (4.16%)	127.486 MHz
Top Level Implementation	**Slice Flip Flops:** 459 (0.961 %) **Total 4 Input LUTs**: 768 (1.61 %) **Occupied Slices:** 502 (2.10 %) **BUFGMUXs (Global Clock):** 1 (4.16%) **DSP48As:** 1 (0.794 %) **RAMB16BWERs:** 2 (1.59 %)	54.383 MHz

Table 4.2 Resurces Consumed by Shared Resource Coarse Frequency Estimation Algorithm Implementation at various levels

4.2.6. Area Consumption and Timing Results for performance optimized design

The area and post place and route timing results for performance based implementation are shown in Table 4.3.

Component	Resources Consumed	Maximum Clock (Post Place and Route)
Data Path (Figure 3.12)	**Slice Flip Flops:** 304 (0.64 %) **Total 4 Input LUTs**: 557 (1.17 %) **Occupied Slices:** 321 (1.34 %) **BUFGMUXs (Global Clock):** 1 (4.16%) **DSP48As:** 6 (4.76 %)	124.486 MHz (with the utilization of all available pipeline registers)
Control Path	**Slice Flip Flops:** 36 (0.07 %) **Total 4 Input LUTs**: 67 (0.14%) **Occupied Slices:** 40 (0.17 %) **BUFGMUXs (Global Clock):** 1 (4.16%)	131.509 MHz
Top Level Implementation	**Slice Flip Flops:** 338 (0.71 %) **Total 4 Input LUTs**: 662 (1.39 %) **Occupied Slices:** 402 (1.68 %) **BUFGMUXs (Global Clock):** 1 (4.16%) **DSP48As:** 6 (4.76 %) **RAMB16BWERs:** 2 (1.59 %)	122.9 MHz

Table 4.3 Resources Consumed by Performance optimized Coarse Frequency Estimation Algorithm Implementation at various levels

4.2.7. Maximum Throughput Calculation for Shared Resource Design

Calculation of maximum throughput for training sequence of length 32 and 64 are shown in eq. 4.2 and eq. 4.3 respectively.

Maximum Acievable Clock of System $= 53.353$ MHz

(Verified with clock constraint 19 ns 50 % duty cycle)

Average Clock Cycles for processing training sequence of length $32 \approx 320$

Average Clock Cycles for processing training sequence of length $64 \approx 570$

$$\text{Throughput for traning sequence of length } 32 = \frac{53.353\text{ M}}{320} = 166.728\text{ Ksps}$$

$$= 166.728\text{ Ksps} \times 32\,\frac{\text{bits}}{\text{sample}} = 5.33\text{ Mbps} \quad \text{----------(4.2)}$$

$$\text{Throughput for traning sequence of length } 64 = \frac{53.353\text{ M}}{570} = 93.601\text{ Ksps}$$

$$= \mathbf{93.601\ Ksps} \times \mathbf{32}\,\frac{\mathbf{bits}}{\mathbf{sample}} = \mathbf{2.99\ Mbps} \quad \text{----------(4.3)}$$

4.2.8. Maximum Throughput Calculation for Performance optimized Design

The throughput calculation for performance based for training sequence of length 32 and 64 is shown in eq. 4.4 and eq. 4.5 respectively.

Maximum Acievable Clock of System $= 120.3$ MHz

(Verified with clock constraint 8.5 ns 50 % duty cycle)

Average Clock Cycles for processing training sequence of length $32 \approx 160$

Average Clock Cycles for processing training sequence of length $64 \approx 285$

Throughput for traning sequence of length $32 = \frac{120.3\ \text{M}}{160} = 751.875\ \text{Ksps}$

$$= \mathbf{751.875\ Ksps} \times \mathbf{32}\,\frac{\mathbf{bits}}{\mathbf{sample}} = \mathbf{24.06\ Mbps} \quad \text{----------(4.4)}$$

Throughput for traning sequence of length $64 = \frac{120.3\ \text{M}}{320} = 422.10\ \text{Ksps}$

$$= \mathbf{422.10\ Ksps} \times \mathbf{32}\,\frac{\mathbf{bits}}{\mathbf{sample}} = \mathbf{13.51Mbps} \quad \text{----------(4.5)}$$

4.2.9. Simulated Waveforms

Figure 4.13 shows simulated waveform of Coarse Frequency Estimation Algorithm Implementation for training sequence of length 32 for performance optimized design. Similarly, Figure 4.14 shows the simulated waveform for training sequence of length 64 for shared resource design.

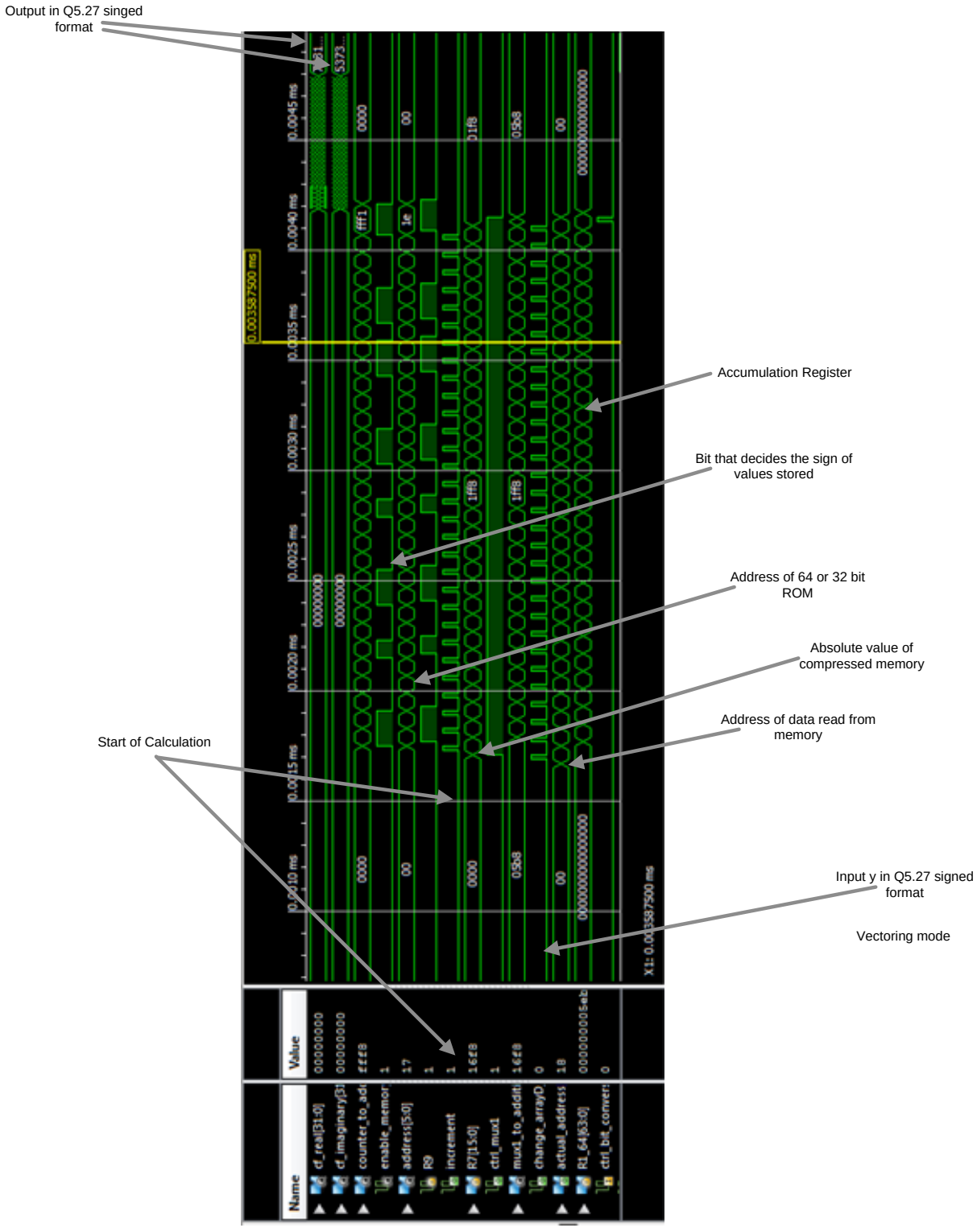

Figure 4.13 Simulated Waveform for Performance Optimized Coarse Frequency Estimation Algorithm Implementation for training sequence length 32

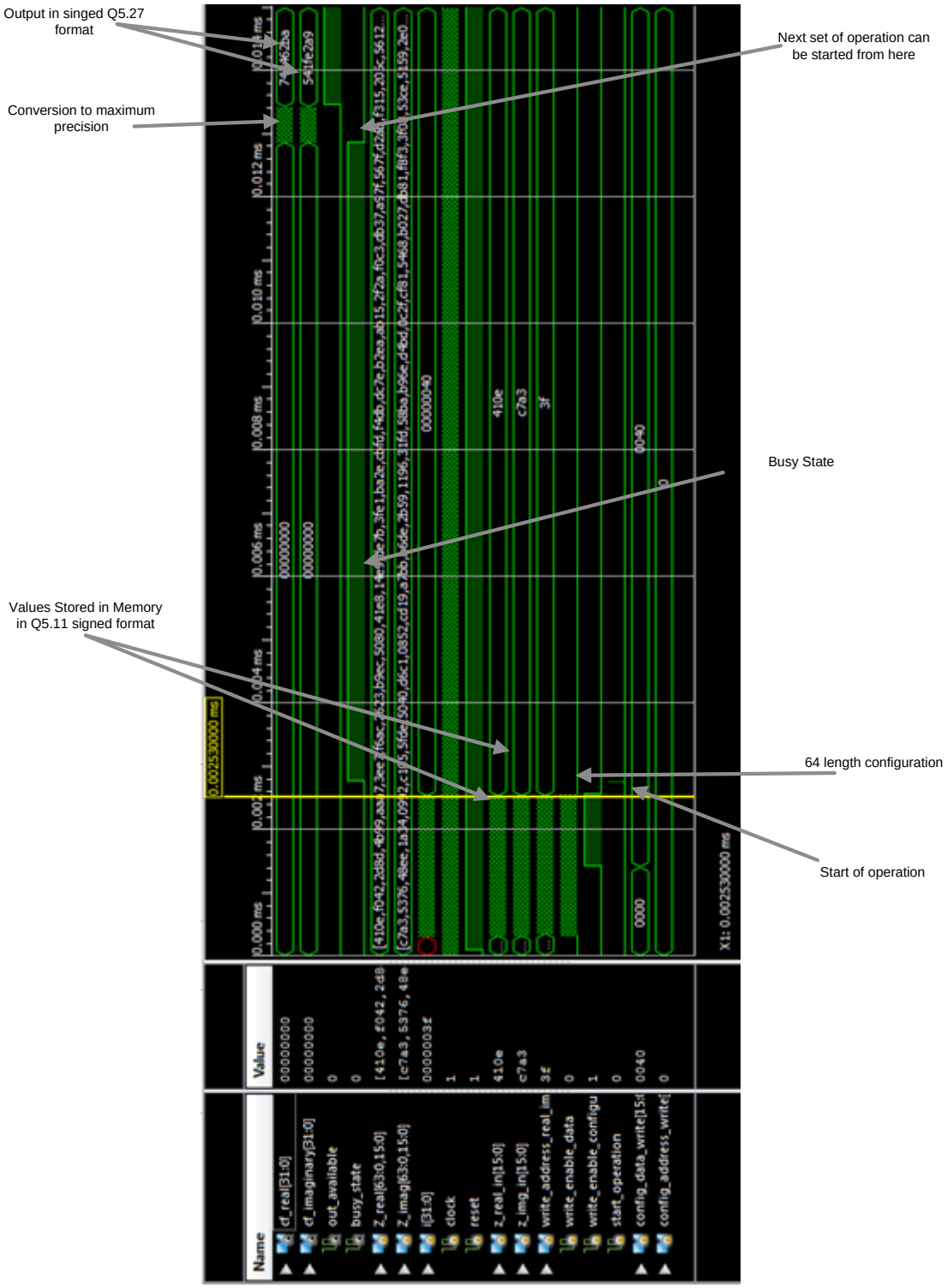

Figure 4.14 Simulated Waveform for Shared Resource Coarse Frequency Estimation Algorithm Implementation for training sequence length 64

4.2.10. Comparison with Previous Implementations

Detailed information about previous implementations is given in 2.6.4.1. Most of the implemented architectures are requirements based. Current implementation uses only one DSP block for its implementation. Overall shared nature of architecture makes it area optimized. Moreover, the compression of lookup table has been performed such that memory requirements reduce from 1.5 kb to only 96 bits.

4.3 Results Discussion

The basic idea behind presenting two architectures for coarse frequency estimation is to provide comparison of various attributes that effect throughput and area utilization The two different implementations of same algorithm provide insight of the tradeoffs between the parameters i-e by comparing the Table 4.2 and Table 4.3, it can be concluded if used properly, dedicated resources provide boost in throughput. Pipelining also plays an important role of increasing the throughput in performance optimized design. This design contains three pipeline stages that boost its clock speed from 43 MHz to 120 MHz.

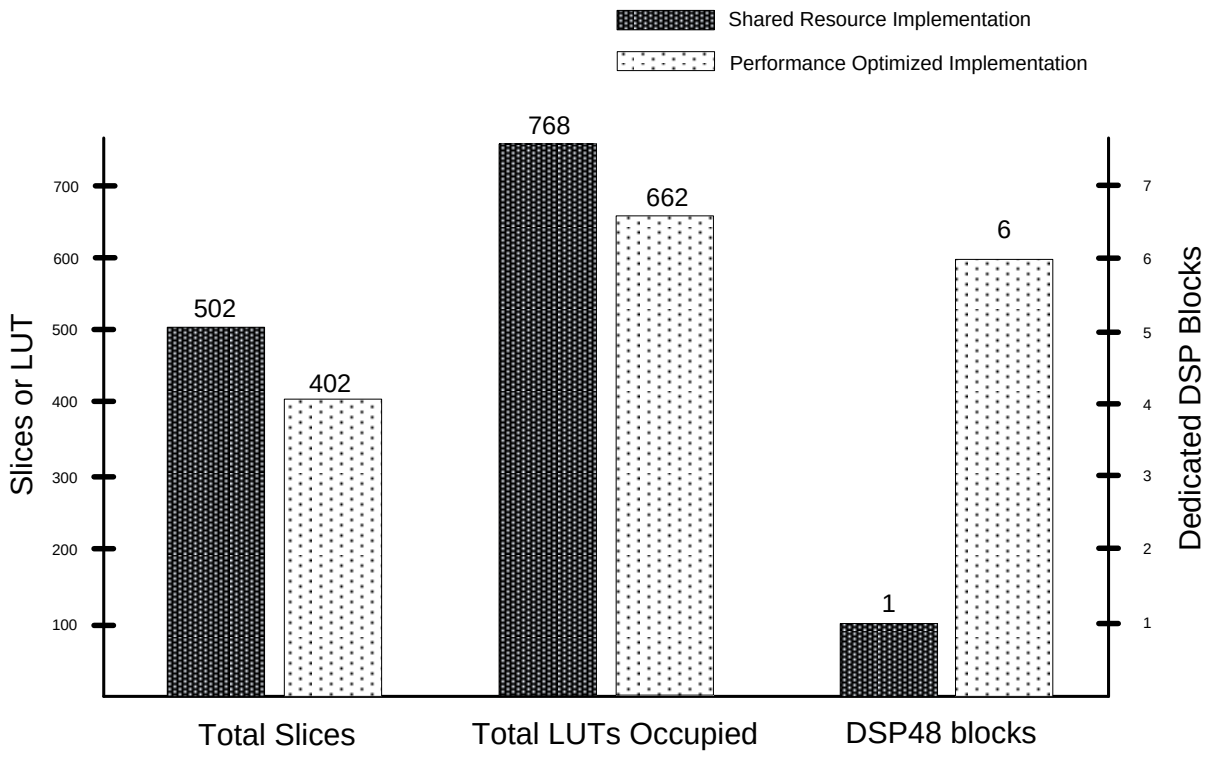

Figure 4.15 Slice Flip Flop and DSP48 blocks Utilization Comparison of Coarse Frequency Estimation Implementation

The comparison shown in Figure 4.15 and Figure 4.16 tells the complete picture of the tradeoff between performance and utilization of dedicated DSP blocks. Throughput of shared resource implementation is low because it requires more clock cycles for complex multiplication. For high throughput design, complex multiplication is done in multiple dedicated hardware multipliers.

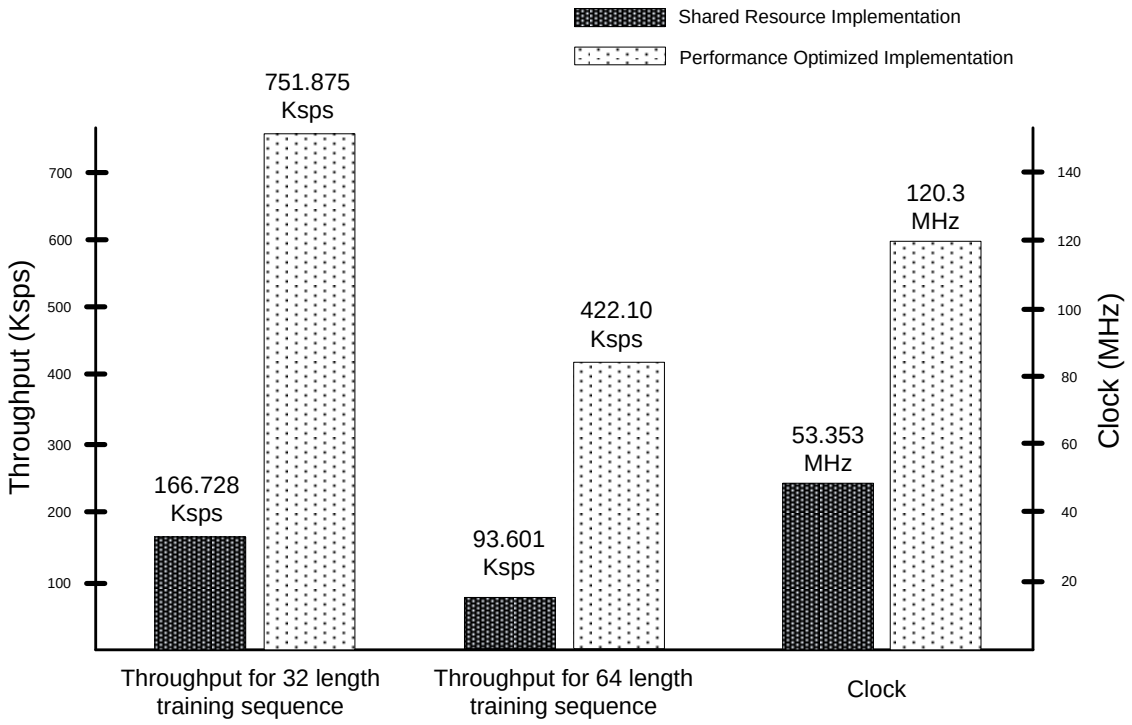

Figure 4.16 Throughput and Clock for both designs of Coarse Frequency Estimation Implementation

The CORDIC implementation is used to calculate the final frequency estimation factor. This implementation is done with iterative architecture. This is due to the fact that the input and output interface of CORDIC core implementation operates iteratively. Implementing CORDIC core in unrolled form can decrease the efficiency of design for the values for which CORDIC algorithm do not converge. To get higher throughputs, multiple modules of CORDIC implementation can be used.

Another important factor considered while designing CORDIC architecture is to facilitate other algorithms that require sine and cosine calculation in the configuration known as rotation mode. Coarse Frequency Estimation requires the angle calculation only which has also been integrated in its design. Figure 4.17and Figure 4.18 illustrates area utilization and throughput respectively.

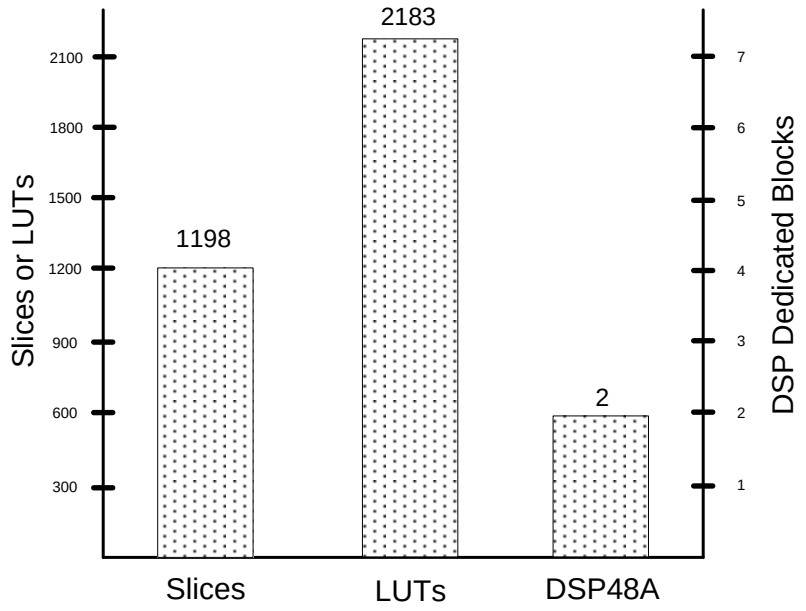

Figure 4.17 Area Utilization of CORDIC Top Level Implementation

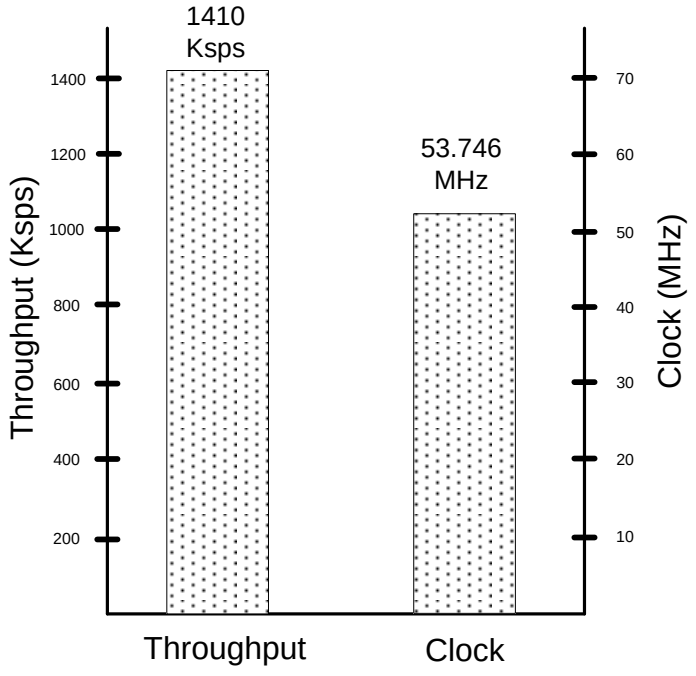

Figure 4.18 Clock and Throughput of CORDIC Top Level Implementation

The bottle neck for CORDIC implementation is the implementation of trigonometric identities and the iteration of subtraction or addition that brings input value to the valid range of CORDIC

core. By using this CORDIC implementation with both designs of Coarse Frequency Estimation the final throughput is shown in eq. 4.6 to eq. 4.9 and illustrated by Figure 4.19.

32 Length − Performance Optimized Implementation throughput with current CORDIC

$$= \frac{1}{\frac{1}{\text{Throughput of Performance Opt. Imp for 32 length training}} . + \frac{1}{\text{Throughput of CORDIC}}}$$

$$= \frac{\mathbf{1}}{\mathbf{1.33\ us + 0.71\ us}} = \mathbf{490.196\ Ksps} \text{ - - - - - - - - - - } \mathbf{(\ 4.6\ )}$$

64 Length − Performance Optimized Implementation throughput with current CORDIC

$$= \frac{1}{\frac{1}{\text{Throughput of Performance Opt. Imp for 64 length training}} . + \frac{1}{\text{Throughput of CORDIC}}}$$

$$= \frac{\mathbf{1}}{\mathbf{2.37\ us + 0.71\ us}} = \mathbf{324.77\ Ksps} \text{ - - - - - - - - - - -} \mathbf{(\ 4.7\ )}$$

32 Length − Shared Resource Implementation throughput with current CORDIC

$$= \frac{1}{\frac{1}{\text{Throughput of Performance Opt. Imp for 32 length training}} . + \frac{1}{\text{Throughput of CORDIC}}}$$

$$= \frac{\mathbf{1}}{\mathbf{6.00\ us + 0.71\ us}} = \mathbf{149.031\ Ksps} \text{ - - - - - - - - - - } \mathbf{(\ 4.8\ )}$$

64 Length − Shared Resource Implementation throughput with current CORDIC

$$= \frac{1}{\frac{1}{\text{Throughput of Performance Opt. Imp for 64 length training}} . + \frac{1}{\text{Throughput of CORDIC}}}$$

$$= \frac{\mathbf{1}}{\mathbf{10.68\ us + 0.71\ us}} = \mathbf{87.796\ Ksps} \text{ - - - - - - - - - - -} \mathbf{(\ 4.9\ )}$$

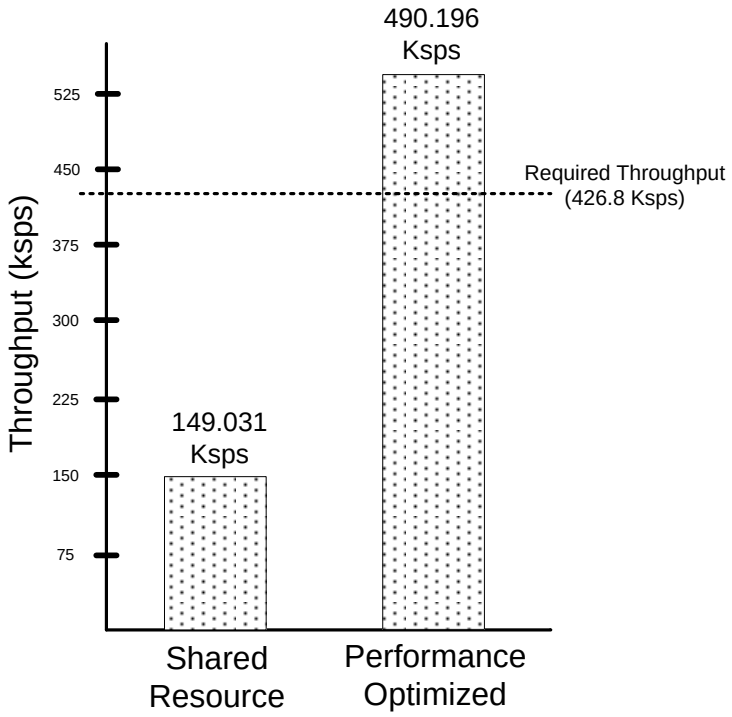

Figure 4.19 Final Throughput for 32 length training sequence

From all the discussion above, it can be concluded that, the two presented implementations can be divided into two categories.

- Design with throughput upto 140 Ksps
- Design with throughput upto 490 Ksps

As the requirement is 426.8 Ksps, so the second design can be used with in SDR of final throughput of 384 kbps.

4.4 Clock Relationship

In order to accomplish coarse frequency estimation operation, the cascade operation of any of the coarse frequency architecture and CORDIC is required. As evident by the results, both of these architectures operate in different clocks so in order to transfer the data from one module to another module, some interface is required. This interface will ensure the smooth acquisition of data from coarse frequency estimation to CORDIC module.

The interface between both modules can simply be achieved by using a register that is operating at CORDIC clock frequency. Once the data is available at the output of coarse frequency estimation architecture, it can be written to this intermediate memory and coarse frequency can be used for next operation immediately. Meanwhile, the CORDIC can start its operation by reading the input data from intermediate memory.

While designing both of these architectures, it has been ensured that data should never get over run. It has been ensured by designing CORDIC such that its throughput remains higher then fastest coarse frequency estimation implementation. In this way, CORDIC completes processing the current output data before coarse frequency estimation can generate next data of concern. CORDIC implementation gets in idle state if it does not get the data in the next processing cycle. So the data under run condition is also handled successfully without the generation of any exception.

4.5 Summary

This chapter gives the results of the implementations of designs given in previous chapter. It includes two coarse frequency estimation designs and CORDIC implementation. CORDIC is used to calculate the angle of estimation after the complex number is provided by the Coarse frequency estimation architecture.

Chapter 5: Conclusion and Future Work

5.1 Conclusion

The primary objective of this thesis is to propose the design and implementation of Coarse Frequency Estimation Algorithm for SDR receiver of throughput 384 Ksps. The approach to the solution of the problem is to design two different architectures. One of them is shared resource design while other is performance optimized. The major characteristic that differs between both is the utilization of dedicated resources.

The performance optimized design explicitly uses dedicated multipliers in hardware while shared resource design uses only one dedicated hardware multiplier. As a consequence, it becomes evident that the use of dedicated hardware helps in increasing the throughput of the system. Moreover, performance optimized design also utilizes pipeline stages.

After the careful examination of Coarse Frequency Estimation Algorithm, it can be divided into two parts. One part can be named as pre calculable part and other as real time calculation part. The pre calculable part can be implemented such that it consumes only 96 bit ROM and few resources for counter as compare to the direct implementation which needs at least 1.5 kb of ROM. Its input is the received training sequence of either 32 or 64 bit length depending upon the mode of operation. The output of this module is given to the input of CORDIC module that calculates the final Coarse Frequency Offset in the received burst. The format of implementation is Q5.11 signed. Both of the implemented designs utilize this compression of pre calculable part.

One of the most important parts is the design and implementation of CORDIC algorithm which will be used for calculating fundamental trigonometry functions. It consists of two parts; core and interface. Due to the limitation in convergence of CORDIC algorithm, the interface provides the behavior of trigonometric identities to facilitate the calculation for wide range of inputs. The overall design of core is of 32 bit iterative nature in Q5.11 signed format. Moreover, in order to utilize resources in better way, both vector and rotation modes of CORDIC Algorithm are

implemented in single system such that it has the capability to change modes in runtime. This CORDIC implementation

Literature review gives extensive relevant information on both digital signal processing and digital system design. It includes concepts related to modulation, demodulation, multiple access techniques and many others. Concepts related to architecture and state machines are also discussed. All of the algorithms of interest along with relevant previous implementations are presented in detail.

For each of the implementation, mainly the benchmark criteria are the percentage error, throughput and consumption of resources. In order to calculate percentage error, comparison of implemented system is done with MATLAB. Visual Comparisons of graph shows the calculated values in hardware along with MATLAB. Objective of these graphs are to give the rough idea of range in which the calculated output lies. Some of the simulation waveforms provide the visualization of some scenarios in which these systems can perform. The model of FPGA is from the family of XLINX 3A- DSP xc3sd3400a. Based on the throughput results achieved, it has been discussed that, the performance optimized design can be used for SDR implementation of 384 Ksps.

5.2 Future Work

Given in Figure 2.14, besides the implementation of CORDIC algorithm and Coarse Frequency estimation algorithm implementation of channel estimation, fine frequency estimation and residual phase estimation can be done on FPGA. Most of these implementations can use CORDIC implementation to figure out their respective trigonometric calculations. In order to achieve desired throughput with current implemented CORDIC, it is recommended to use its multiple instances in parallel.

For coarse frequency estimation algorithm, the offline calculable part can be saved outside of FPGA chip and can be calculated in general purpose embedded system processor. This way not only saves area on FPGA but also introduces further factor of flexibility. Moreover, the module of coarse frequency estimation can be interfaced with the software. This interfacing can be

realized by first connecting the design with Microblaze processor and then communicating with microcontroller or DSP.

5.2.1. Channel Estimation

Estimating the channel is actually a measurement of its frequency response. This response is equalized using the method known as channel equalization. In this method, received samples are filtered through inverted channel response to compensate unwanted suppression caused by channel. But before this, the estimation of channel response is required to be made. This estimation is known as channel estimation. Suitable algorithm [7] that can be used for channel estimation is explained below.

Let there are a total of N received samples. If these samples are received after passing through the channel whose instantaneous gain is $h_0, h_1, h_2 \; to \; h_{k-1}$ then let's define the following,

$$\mathbf{r} = [\mathbf{r(0)}\ \mathbf{r(1)}\ \mathbf{r(2)} \ldots\ \mathbf{r(N-1)}]^{\mathbf{T}} \text{ - - - - - - - - - - (5.1)}$$

$$\mathbf{h} = [\ \mathbf{h_0}, \mathbf{h_1}, \mathbf{h_2} \ldots\ \mathbf{h_{K-1}}]^{\mathbf{T}} \text{ - - - - - - - - - - - - - (5.2)}$$

$$\mathbf{W(v)} = \mathbf{diag}\{\mathbf{1}, \mathbf{e}^{\frac{\mathbf{j2\pi v}}{\mathbf{N}}}, \mathbf{e}^{\frac{\mathbf{j2\pi 2v}}{\mathbf{N}}}, \mathbf{e}^{\frac{\mathbf{j2\pi 3v}}{\mathbf{N}}}, \ldots, \mathbf{e}^{\frac{\mathbf{j2\pi(N-1)v}}{\mathbf{N}}}\} \text{ - - - - - - - - - (5.3)}$$

where $'v'$is the normailized frequency offset & $'K'$is total no. of path gains in channel

Also the noise sample array for N samples are defined in eq. 5.4 and transmitted training symbol matrix can be defined as eq. 5.5 where transmitted training symbols can be denoted by **s.**

$$\mathbf{n} = [\mathbf{n(0)}\ \mathbf{n(1)}\ \mathbf{n(2)} \ldots\ \mathbf{n(N-1)}]^{\mathbf{T}} \text{ - - - - - - - - - - (5.4)}$$

$$\mathbf{S} = \begin{bmatrix} \mathbf{s(0)} & \mathbf{s(-1)} & \cdots & \mathbf{s(-K+1)} \\ \mathbf{s(1)} & \mathbf{s(0)} & \cdots & \mathbf{s(-K+2)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{s(N-1)} & \mathbf{s(N-2)} & \cdots & \mathbf{s(-K+N)} \end{bmatrix} \text{ - - - - - - - - (5.5)}$$

$where\ s(v) = 0\ for\ v < 0$

The received samples can also be represented as

$$\mathbf{r} = \mathbf{e}^{\mathbf{j\emptyset}}.\mathbf{W(v)}.\mathbf{S}.\mathbf{h} + \mathbf{n} \text{ - - - - - - - - (5.6)}$$

On the basis of all equations given above, maximum likelihood channel estimation can be represented as eq. 5.7.

$$\hat{\mathbf{h}} = [\mathbf{S^H}.\mathbf{S}]^{-1}\mathbf{S^H}.\mathbf{W^H}(\hat{\mathbf{v}}).\mathbf{r} \text{ - - - - - - - - } \mathbf{(5.7)}$$

The final channel compensation of received samples can be represented as,

$$\boldsymbol{r'} = \boldsymbol{r}\ \{\overline{(\hat{\boldsymbol{h}}_{\mathbf{0}}) + (\hat{\boldsymbol{h}}_{\mathbf{1}}) + (\hat{\boldsymbol{h}}_{\mathbf{2}}) + \cdots + (\hat{\boldsymbol{h}}_{\boldsymbol{K-1}})}\} \text{ - - - - - - - - } \mathbf{(5.8)}$$

where $\mathbf{r'}$ is the channel compensated received samples vector.

The implementation of the given algorithm of channel estimation can be implemented for 384 Ksps. If carefully examined, the matrix W and **S** given in eq. 5.3 and eq. 5.5 are offline calculable and the result can be converted from matrix to array with minimum precision sacrifice. This is due to the fact that the matrix $\boldsymbol{S}$ remains constant for any given length of training sequence. If algorithmic decision is to estimate the channel up to four coefficients then there can only be two matrices of $\boldsymbol{S}$**;** for training sequence of 32 and 64 lengths of order 32 by 4 and 64 by 4 respectively. Hence the order of result of offline part $[\boldsymbol{S}^H.\boldsymbol{S}]^{-1}\boldsymbol{S}^H$ is either 4 by 32 or 4 by 64.

A term $\boldsymbol{W}(\hat{v})$ is a diagonal matrix of either 32 by 32 or 64 by 64 (depending upon length of training sequence) order with its entries only calculable with appearance of real time received training symbols. But before implementation, by further observation, it can be proved that the factor $[\boldsymbol{S}^H.\boldsymbol{S}]^{-1}\boldsymbol{S}^H.\boldsymbol{W}^H(\hat{v})$ can be calculated without matrix multiplication.

The order of result in eq. 5.7 will always be 4 by 1; 4 coefficients of channel estimation. But as shown in eq. 5.8, the channel compensation is done by multiplying the accumulation of channel estimation coefficients with received data (equalization using FIR filter). Hence, a single summed value of channel coefficient is enough for channel compensation. Keeping this fact in mind with the discussion done in previous paragraph, following steps can be done in order to design channel estimation algorithm.

- Add all rows of matrix $[\boldsymbol{S}^H.\boldsymbol{S}]^{-1}\boldsymbol{S}^H$ to an array.
- Convert $\boldsymbol{W}^H(\hat{v})$ to an array too by picking all diagonal elements of matrix.

- Both $[\boldsymbol{S}^H.\boldsymbol{S}]^{-1}\boldsymbol{S}^H$ and $\boldsymbol{W}^H(\hat{v})$ are now an array of same order; 1 by 32 or 1 by 64 depending upon training symbols. Multiply the elements of both arrays point by point
- Multiply this result with received training array $\boldsymbol{r}$. The result will be single channel estimation coefficient that will be the sum of 4 channel coefficients of order one by one.

For the transmitted training sequence shown in Figure 3.6, the value of matrix converted array $[\boldsymbol{S}^H.\boldsymbol{S}]^{-1}\boldsymbol{S}^H$ is shown in Figure 5.1.

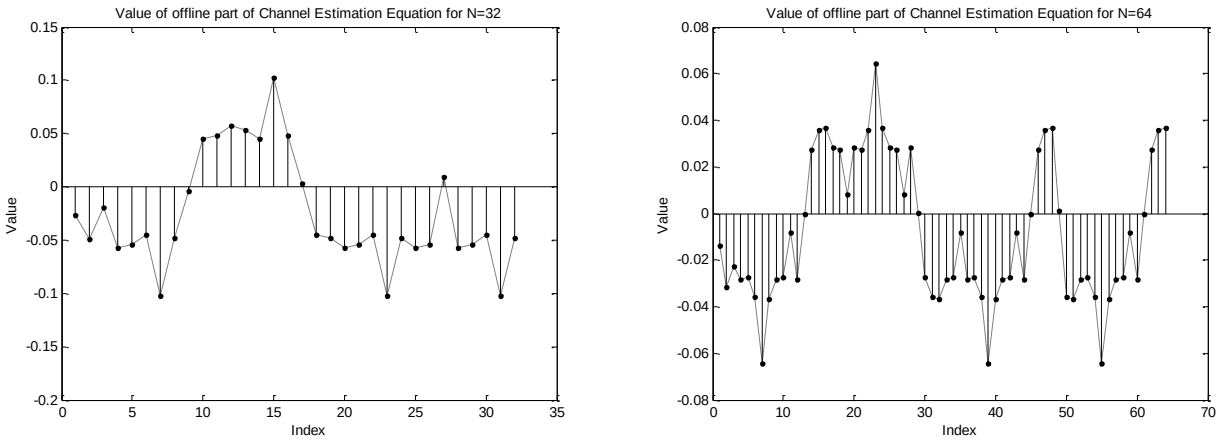

Figure 5.1 Offline calculable part of Channel Estimation Equation (summed up as an array)

The implementation of this algorithm becomes straight forward because after simplification, three arrays are required to be point by point multiplied. So, effectively the datapath requires the resources to multiply complex numbers and memory to retain the data. Offline calculable $[\boldsymbol{S}^H.\boldsymbol{S}]^{-1}\boldsymbol{S}^H$, will be saved in ROM separately for both N=32 and N=64. While the members of factor $\boldsymbol{W}^H(\hat{v})$ are calculated in rum time and used as per needed. The received sequence will be saved in RAM.

5.2.2. Fine Frequency Estimation

The fine frequency estimation and compensation algorithm is applied once coarse frequency algorithm is done. It estimates the residual frequency within acceptable range but much more complex and computationally high profile. Suitable algorithm for implementation has been implemented in [39].

This algorithm basically measures the phase in the received burst that comes from the remaining offset in the frequency. Moreover, offset for each sample in received data burst R is calculated and compensated separately. Also, we have to append Q_p number of zeros before and after the received burst of length D. Then the estimation can be done by the following equation.

$$\widehat{\boldsymbol{\theta}}(\mathbf{i}-\mathbf{Q_p}) = \left[\frac{\mathbf{1}}{\mathbf{M}}\mathbf{arg}\left\{\sum_{\mathbf{z=max(i-Q_p,\,Q_p+1)}}^{\mathbf{min(i+Q_p-1,D+Q_p)}} \mathbf{e^{jMarg(R(z))}}\right\}\right]_{\frac{2\pi}{\mathbf{M}}} \text{----------(5.9)}$$

$$\mathrm{i} = \mathrm{Q_p} + 1, \mathrm{Q_p} + 2, \ldots, \mathrm{Q_p} + \mathrm{D}$$

where $[]_{\frac{2\pi}{M}}$ denotes modulo $\frac{2\pi}{\mathrm{M}}$ operation & M is the modulation index of PSK modulation

After the fine frequency estimation is done, we will get an array of length D. This array is finally used to compensate the data burst.

$$\mathbf{Compensated_{Data(z)}} = \mathbf{Uncompensated_{Data(z)}}.\mathbf{e^{-i\widehat{\theta}(z)}} \text{ ----------(5.10)}$$

where $\mathrm{z} = 0$ to $\mathrm{D} - 1$

5.2.3. Residual Phase Estimation

If some residual phase still remains after demodulation, this phase estimation algorithm is used as a solution. This algorithm finds out the residual phase (if any) in the demodulated signal r_{de}. Implemented algorithm of this dissertation is shown in the one line equation below. Note that, it is just the multiplication and accumulation of corresponding training samples of received demodulated and transmitted signal.

$$\boldsymbol{\vartheta_{res}} = \mathbf{arg}\left\{\sum_{x=0}^{N-1} \boldsymbol{a(x)r_{de}(x)}\right\} \text{ ----------(5.11)}$$

where N is the total number of training samples & a is training sequence to be transmitted

The abstract top level diagram of the top level diagram that contains CORDIC module, Coarse Frequency Estimation, Channel Estimation, Fine frequency estimation and residual phase estimation is shown in

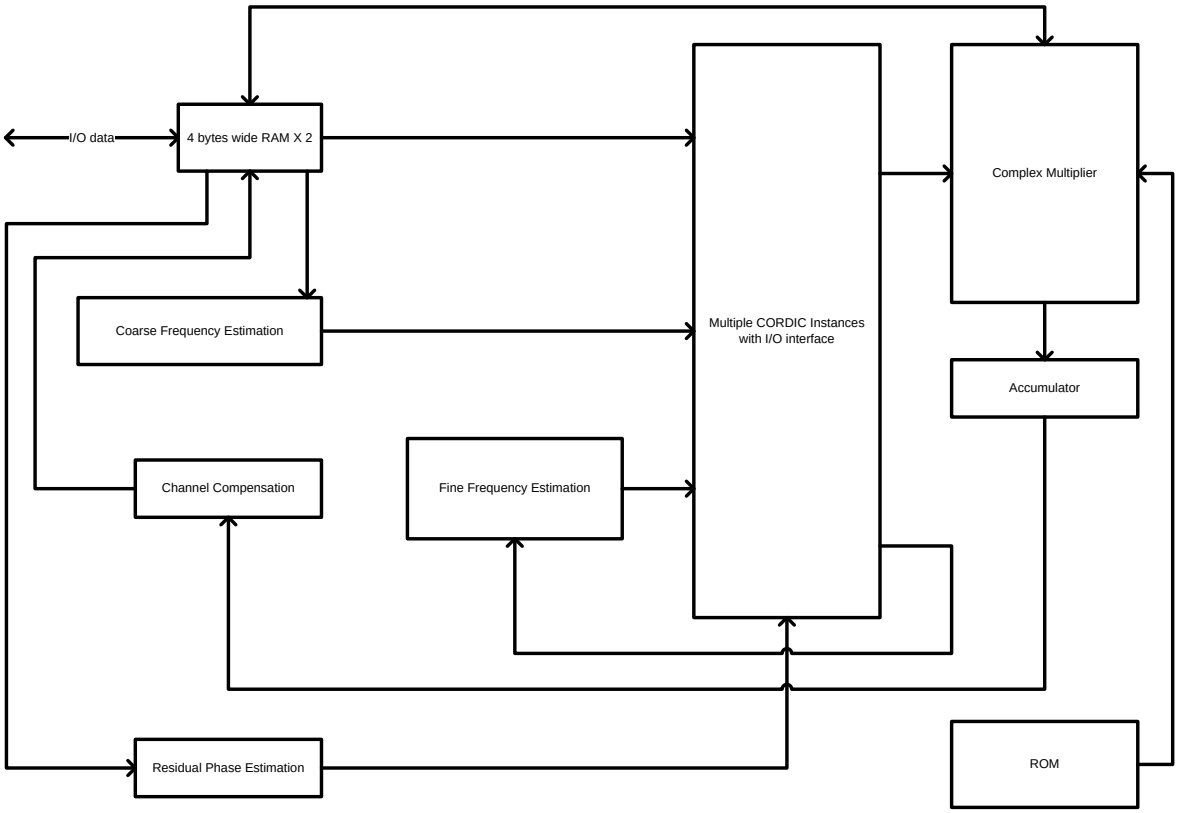

Figure 5.2 Abstract Diagram of Top Level Datapath Implementation

References

[1] B.P. Lathi, *Modern Digital and Analog Communication Systems*, 3rd ed. New York: Oxford University Press, 1998.

[2] William Stallings, *Computer Organization & Architecture: Designing for Performance*, 6th ed.: Pearson Education.

[3] M.Morris Mano, *Computer System Architecture*, 3rd ed. New Jersey, USA: Prentice Hall, 1993.

[4] John McAlister, Gaye Lightbody, Ying Yi Roger Woods, *FPGA based implementation of signal processing systems*.: John Wiley & Sons, 2008.

[5] Dr. Shoab Ahmed Khan, *Digital Design of Signal Processing Systems: A Practical Approach*.: John Wiley & Sons, Ltd, 2011.

[6] Bernard Sklar, *Digital Communications Fundamental and Applications*, 2nd ed. Los Angeles, California: Prentice Hall.

[7] Hlaing Minn, Vijay K. Bhargava, Khaled Ben Letaief, "A Robust Timing and Frequency Synchronization for OFDM systems," *IEEE Transaction on Wireless Communication*, vol. 2, no. 4, pp. 822-838, July 2003.

[8] Alan V. Willsky, S Hamid Nawab Alan V. Openheim, *Signals And Systems*, 2nd ed., Alan V. Openheim, Ed. New Jersey: Prentice Hall.

[9] M. Moeneclaey, S. A. Fechtel H. Meyr, *Digital Communication Receivers; Synchronizaton, Channel Estimation and Signal Processing*. New York, US: John Wiley and Sons, 1998.

[10] Dimitris G. Manolakis John G. Proakis, *Digital Signal Processing: Principles, Algorithms and Applications*, 4th ed. Boston, MA, US: Pearson Prentice Hall, 2007.

[11] H. Nyquist, "Certain Topics in Telegraph Transmission Theory," *AIEE Transactions*, vol. 47, no. 2, pp. 617-644, April 1928.

[12] W. Effing W. Rankl, *Smart Card Handbook*, 3rd ed.: John Wiley and Sons, Limited, 2003.

[13] David L. Adamy, *EW 102: A Second Course in Electronic Warfare*. London, United Kingdom: Horizon House Publications, 2004.

[14] Rodger E. Ziemer, *Fundamentals of Spread Spectrum Modulation*, Virginia Tech William Tranter, Ed. USA: Morgan and Claypool, 2007.

[15] Spartan-3A DSP FPGA Family Data Sheet, October 4, 2010.

[16] Ian Elliott Peter Minns, *FSM-based Digital Design using Verilog H DL*. England: John Wiley & Sons, 2008.

[17] Austin Lesea, Rene Ritcher Doug Amos, *FPGA-based Prototyping Methodology Manual: Best Practices in Design-for-Prototyping*. Mountain View, California, USA: Synopsys, 2011.

[18] Jack E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Transaction on Electronic Computer*, vol. EC-8, no. 3, pp. 330-334, September 1959.

[19] J. S. Walther, "Unified Algorithm for elementary functions," in *spring joint computer conference AFIPS '71*, New York, 1971, pp. 379-385.

[20] Jean-Michel Muller, *Elementary Functions Algorithms and Implementations*, 2nd ed. NewYork, USA: Birkhauser, 2006.

[21] Ray Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *FPGA '98 Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field*

programmable gate arrays, New York, 1998, pp. 191-200.

[22] Yu Hen Hu, "The Quantization effects of CORDIC Algorithm," *IEEE TRANSACTIONS ON SIGNAL PROCESSING*, vol. 40, no. 4, pp. 834-844, Apr 1992.

[23] R. Cumplido, M. Arias E. O. Garcia, "Pipelined CORDIC Design on FPGA for a Digital Sine and Cosine Waves," in *3rd International Conference on Electrical and Electronics Engineering*, Puebla, Mexico, 2006, pp. 1-4.

[24] A. Boudabous, W. Kharrat, N. Masmoudi N. Neji, "Architecture and FPGA Implementation of the CORDIC Algorithm for Fingerprints Recognition Systems," in *2011 8th International Multi-Conference on Systems, Signals & Devices*, March 2011, pp. 1-5.

[25] Jean-Michel Muller Jean Duprat, "The CORDIC Algorithm: New Results for Fast VLSI Implementation," *IEEE TRANSACTIONS ON COMPUTERS*, vol. 42, no. 2, pp. 168-178, Feb 1993.

[26] M. D. Ercegovac and T. Lang, "Redundant and On-Line CORDIC : Application to Matrix Triangularization and SVD," *IEEE Transactions on Computers*, vol. 39, no. 6, pp. 725-740, June 1990.

[27] Randy Saut Purba Trio Adiono, "Scalable Pipelined CORDIC Architecture Design and Implementation in FPGA," in *International Conference on Electrical Engineering and Informatics*, Selangor, Malaysia, 2009, pp. 646-649.

[28] P.Muralidhar S. Bhuria, "FPGA Implementation of Sine and Cosine Value Generators using Cordic Algorithm for Satellite Attitude Determination and Calculators," in *International Conference on Power, Control and Embedded Systems (ICPCES), 2010* , Warangal, India, 2010, pp. 1-5.

[29] Min Ye, "FPGA Implementation of CORDIC-Based Square Root Operation for Parameter Extraction of Digital Pre-Distortion for Power Amplifiers," in *6th International Conference on Wireless Communications Networking and Mobile Computing*, Ningbo, China , 2010, pp.

1-4.

[30] H. Meyr and P. Sehier F. Classen, "Maximum likelihood open loop carrier synchronizer for digital radio," in *IEEE International Conference on Communications*, Geneva, 1993.

[31] J. h. Ge, Y. Wang B. Ai, "Frequency offset estimation for OFDM in wireless communications," *IEEE Transactions on Consumer Electronics*, vol. 50, no. 1, pp. 73-77, 2004.

[32] M. Luise and R. Reggiannini, "Carrier Frequency Recovery in All digital modems for burst mode transmissions," *IEEE Transactions on Communications*, vol. 43, no. 234, pp. 1169-1178, 1995.

[33] U. Mengali and M. Morelli, "Data-Aided Frequency Estimation for Burst Digital Transmission," *IEEE Transactions on Communications*, vol. 45, no. 1, pp. 23-25, Jan 1997.

[34] M. P. Fitz, "Planar Filter Techniques for Burst mode Carrier Synchronization," in *Global Telecommunications Conference*, West Lafayette, IN, US, 1991.

[35] Hyoung Jin Yun, Myung Hoon Sunwoo, Pansoo Kim, and Dae-Ig Chang Jang Woong Park, "Efficient Coarse Frequency Synchronizer Using Serial Correlator for DVB-S2," in *IEEE Internatinal Symposium on Circuits and Systems*, Suwon, South Korea, May 2008.

[36] Zaiwang Dung Linghi Huang, "The implementation of estimation and correction of carrier frequency offset of COFDM system in DAB receiver ," in *5th international conference on ASIC*, Beijing, 2003.

[37] J Singh and M Faulkner K Wang, "FPGA Implementation of an OFDM-WLAN Synchronizer," in *Second IEEE International Workshop on Electronic Design, Test and Application*, Melbourne, 2004.

[38] Geert Vanwijnsberghe, Peter Van Wesemael, Tom Huybrechts, Steven Thoen MaryseWouters, "Real Time Implementation on FPGA of an OFDM based Wireless LAN

modem extended with Adaptive Loading," in *Proceedings of the 28th European Solid State Circuits Conference*, Heverlee, Belgium, 2002.

[39] A. M. Viterbi A. J. Viterbi, "Nonlinear estimation of PSK-modulated carrier phase with application to burst digital transmission," *IEEE Transactions on Information Theory*, vol. 29, no. 4, pp. 543-551, Jul 1983.

[40] Mohamed M. Khairy, H. A. H. Fahmy, S. E.D. Habib K. ElWazeer, "FPGA Implementation of an Improved Channel Estimation Algorithm for Mobile WiMAX," in *International Conference on Microelectronics*, Cairo, 2009, pp. 280-283.

[41] T. Ogunfunmi J. S. Park, "FPGA implementation of channel estimation for MIMO-OFDM," in *IEEE International Symposium on Circuits and Systems*, Santa Clara, 2011, pp. 705-708.

[42] Farhad B. Verahrami, Ajit Shenoy Harvey Chalmers, "Digitally Implemented Fast Frequency Estimator/ Demodulator for Low Bit Rate Maritime and mobile data communications without the use of an acquisition preamble," 5272446, Dec 21, 1993.

[43] Vijay K. Bhargava, Khaled Ben Letaief Hlaing Minn, "A Robust Timing and Frequency Synchronization for OFDM systems," *IEEE Transaction on Wireless Communication*, vol. 2, no. 4, pp. 822-838, July 2003.

[44] Ray Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *FPGA '98 Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*, New York, 1998, pp. 191-200.

Proposed Solution Specifications

Product	Coarse Frequency Estimation	
Architecture	Dedicated Resource Pipeline Based	Shared Resource
Maximum Clock	120.3 MHz	53.3 MHz
Number of DSP48 Blocks Used	6	1
Pipeline Stages	3	Pipelining is not used
Maximum Throughput (for 32 length training sequence)	751.875 Ksps	166.728 Ksps
Maximum Throughput (for 64 length training sequence)	422.10 Ksps	93.601 Ksps
Input Format	Q5.11 signed	
Output Format	Q5.27 signed	
Format of Operation	Write training Sequence to the input memory, write length of training sequence in configuration register and trigger start_operation for one clock cycle. In order to get higher throughput, replace input data memory by custom size buffer so that the time required to write data to memory can be reduced.	
Output Availability Indication	out_available pin goes high	

Product	CORDIC Implementation
Architecture	Iterative based and ROM indexed
Maximum Clock	53.746 MHz
Number of DSP48 Blocks Used	2
Pipeline Stages	Pipelining is not used
Maximum Throughput	1410 Ksps
Modes	**Rotation Mode:** Calculation of sine and cosine function **Vectoring mode:** Calculation of angle of complex number
Input Format	Q5.27 signed
Output Format	Q5.27 signed
Solution of Convergence Problem	Yes. Trigonometric identity interfaces are present at input and output.
Format of Operation	Place the data and mode of operation at the input and then trigger start_calculation; for rotation mode the mode of operation is set by triggering mode_of_calcualtion to low and vice versa for vectoring mode.
Output Availability Indication	Out_ready pin goes high

Appendix: Abbreviations

AFE	Analog Front End
AM	Amplitude Modulation
ASIC	Application Specific Integrated Circuit
BER	Bit Error Rate
CDMA	Code Division Multiple Access
CLB	Configurable Logic Block
CORDIC	Coordinate Rotation Digital Computer
DCM	Digital Clock Manager
DSSS	Direct Sequence Spread Spectrum
FDMA	Frequency Division Multiple Access
FPGA	Field Programmable Gate Array
FHSS	Frequency Hopping Spread Spectrum
FM	Frequency Modulation
FSM	Finite State Machines
ISI	Intersymbol Interference
KBps	Kilobytes per Second
Kbps	Kilobits per Second
Ksps	Kilo Samples Per Second
LE	Logic Elements
LUT	Look up Table
Mcps	Mega Chips Per Second
ML	Maximum Likelihood
MMSE	Minimum Mean Square Error
QPSK	Quadrature Phase Shift Keying
SDMA	Space Division Multiple Access

SDR	Software Defined Radio
SNR	Signal To Noise Ratio
SOC	System On Chip
TDMA	Time Division Multiple Access
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
WBNR	Wideband Networking Radio
WCDMA	Wideband Code Division Multiple Access